Mini-Curso

Datacar 00 - Aprenda a criar um sistema completo utilizando as metodologias de Orientação a Objetos – Parte 3

Clube Delphi +PHP

Ano 8 · Edição 98 · R$9,90

DevMedia

Boas Práticas

Saiba como desenvolver corretamente suas aplicações Web com ASP.NET 2.0

ISSN 1517990-7

NA WEB

Confira no portal ClubeDelphi PLUS 3 vídeo-aulas sobre ASP.NET 2.0 e PHP

Boa Idéia
Aprenda a manipular a porta paralela com o Delphi

Banco de Dados
Conheça uma excelente ferramenta para gerenciar seu BD, importar e exportar dados e muito mais

Boas Práticas
Confira a novidade do RAD 2007 para documentar código-fonte

ASP.NET
Globalização – traduzindo seu Web Site para múltiplos idiomas

Easy Delphi
Introdução à criação de componentes

PHP
Dicas para garantir a segurança do seu site contra hackers

developers

webdays 2008

100% de aproveitamento!

22 e 23 de agosto – SP

- Conheça as principais novidades das tecnologias **Java e .NET**
- Palestras sobre **Boas práticas e Arquitetura**
- Certificado de Participação
- **100% de aproveitamento** - Apesar das palestras acontecerem simultaneamente, não haverá perda de conteúdo. No dia do evento você receberá um DVD com o conteúdo completo de todas as palestras em formato de vídeo-aula!

São + de 46 horas de conteúdo.

Principais Temas:

Java	.NET	Extras
Ajax	Linq	SOA
Java FX	WPF	WEB 2.0
Grails	Silverlight 2.0	WEB 3.0
Android	ASP.NET 3.5	Arqutetura Web
Groovy	Ajax	Scrum
Jboss Seam	ASP.NET MVC	Microsoft XNA
JSF	Visual Studio 2008	Ruby on Rails
RIA (Flex, etc)	Novidades C# 3.0 e VB.NET 9.0	Domain Driven Design
Best practices em java	Best practices em .NET	Mashups

Maiores informações:
www.devmedia.com.br/webdays2008
evento@devmedia.com.br
(21) 3382-5025

Inscreva-se já!

Realização DevMedia group

Sumário

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Delphi

Boas Práticas Web

08 – POO e Multicamadas
Boas práticas de desenvolvimento com ASP.NET 2.0
[*Guinther Pauli*]

Boa Idéia

18 – Porta Paralela com Delphi
Entendendo e controlando os registros da porta paralela
[*Thiago Batista Limeira*]

Banco de Dados

24 – Ferramenta para vários servidores de banco de dados
Ferramentas para exportação, importação e administração de banco de dados
[*Luciano Pimenta*]

Web Novidades

34 – Globalizando Aplicações
Traduzindo aplicações em ASP.NET
[*Manoel Edésio*]

Boas Práticas Novidades

40 – Saiba como documentar seu código-fonte
Documentando propriedades e métodos com o RAD Studio 2007
[*Rodrigo Otto Mostaert*]

Boas Práticas Mini-Curso e Expert

44 – Sistema DataCar em OO
Aprenda a criar um sistema completo utilizando as metodologias de Orientação a Objetos – Parte 3
[*Paulo Quicoli*]

Easy Delphi

Easy Delphi

54 – Introdução à criação de componentes
Aprenda os conceitos e como construir seu primeiro componente
[*Guinther Pauli*]

PHP

PHP

60 – Segurança na Web com PHP
Confira algumas dicas de segurança para evitar que usuários habilidosos invadam seu Web Site
[*Adriano Santos*]

Você percebeu os nomes ao lado de cada matéria? Eles indicam o que você vai encontrar no artigo – dessa forma, você também pode ter uma idéia geral do que vai encontrar nesta edição como um todo! Os editores trabalham sempre no sentido de fechar a revista seguindo esta definição, para oferecer a você o melhor conteúdo didático!

Confira abaixo a lista com a definição dos tipos de artigo encontrados nesta edição:

[Mão na Massa] Artigos que focam na resolução de problemas - e não na tecnologia. A tecnologia empregada é apenas conseqüência. O artigo parte do pressuposto que o leitor já conhece a tecnologia, mas tem dificuldade de implementá-la em uma aplicação cotidiana. Por exemplo, um artigo que ao invés de "debulhar" todos os métodos e propriedades de um "Recordset", mostre como usar o Recordset de forma inteligente. para criar um carrinho de compras.

[Mini-Curso] Artigo que faz parte de um mini-curso.

[Delphi for PHP] Artigo com foco na versão PHP do Delphi.

[Boas Práticas] A Um dos objetivos da revista é levar para o leitor não somente as melhores técnicas, mas também as melhores práticas – Esse tipo de artigo foca em técnicas que poderão aumentar a qualidade do desenvolvimento de software.

[Web] Artigos sobre ou que envolvam técnicas de desenvolvimento para WEB.

[Expert] Artigo com foco no leitor avançado.

[Easy Delphi] Com foco no desenvolvedor iniciante.

[PHP] Artigo com foco no PHP puro.

[Novidades] Artigos sobre tecnologias de ponta, que ainda não fazem parte do dia a dia do desenvolvedor mas que prometem ser a próxima febre.

[Boa Idéia] A Esse tipo de artigo é quase um artigo "Mão na Massa" - a diferença é que o exemplo do artigo é tão criativo que é considerado uma 'boa idéia', dentro do contexto de desenvolvimento de software.

Clube Delphi +PHP

Ano 8 - 89ª Edição - 2008 - ISSN 1517990-7 **Impresso no Brasil**

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Adriano Santos
adrianosantos@devmedia.com.br

Equipe Editorial
Fabricio Desbessel, Maikel Scheid, Paulo Quicoli, Luciano Pimenta

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Diagramação
Adolfo Sabino
simininu@yahoo.com.br

Capa
Vinicius O. Andrade
viniciusoandrade@gmail.com

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin
www.devmedia.com.br/central/default.asp
(21) 3382-5025

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5025

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

EDITORIAL

Há 8 anos, quando comecei a desenvolver minhas primeiras aplicações para ASP.NET, senti falta de um personagem que fazia parte do meu dia-a-dia no Delphi Win32: DataModules. Criando uma classe que herde de System.Componentmodel.Component é até possível simular um, mas mais adiante descobri que essa era uma prática terrível no desenvolvimento Web. Diferente do modelo Desktop, queries e conexões não devem ser instanciados todos juntos como em um DataModule, mas por demanda, quando for necessário. A solução é criar uma camada separada de acesso a dados, uma classe, que normalmente chamamos de DAL (Data Access Layer), e nessa camada programar com ADO.NET para fazer as operações de manipulação de banco de dados.

Com a chegada do ASP.NET 2.0, vieram o dbxDataSource e SqlDataSource, mas esses controles só servem para pequenas soluções, protótipos, não são utilizados em aplicativos reais. Um dos principais motivos é que eles não separam o código de acesso a dados da interface de usuário, um dos principais pilares das boas práticas de programação.

Nesta edição, trago um artigo completo que mostra como construir uma arquitetura robusta e principalmente correta de acesso a dados em aplicações ASP.NET 2.0. Você irá aprender a construir uma camada distinta para acesso a dados (DAL) e também uma BLL (Business Logic Layer), para concentrar as regras de negócio da aplicação. E mais, aprenderá a criar DTO's (Data Transfer Objects) para trocar dados entre camadas, além de é claro fazer uso de Stored Procedures, peça fundamental quando se fala em Web. Tudo isso é possível graças a um membro especial da família dos DataSources, o ObjectDataSource. E o melhor de tudo, usando poderosos recursos da programação orientada a objetos e desenvolvimento multicamadas. Apenas não confunda o termo multicamadas aqui com o DataSnap, aqui camadas são classes e não processos.

Ainda nesta edição, aprenda com o Thiago a manipular a porta paralela com o Delphi. O Luciano Pimenta desvenda toda uma excelente ferramenta para administrar o seu banco de dados. O Manoel Edésio ensina a globalização em ASP.NET 2.0, para tornar seus aplicativos adaptáveis a múltiplos idiomas. O Otto descobre uma nova funcionalidade interessante no item boas práticas, usando o RAD 2007, para documentar seu código-fonte. O Paulo Quicoli continua o curso que mostra como construir o popular sistema DataCar agora todo orientado a objetos.

Na sessão Easy, em meu outro artigo, falo de um importante fundamento do Delphi desde os seus primórdios: a criação de componentes. E para finalizar, o Adriano apresenta importantes dicas para você tomar cuidado quando for construir seus sites PHP, no que diz respeito à segurança contra hackers.

Abraço e sucesso com ASP.NET 2.0, multicamadas e POO.

Não deixe de conferir nosso novo BLOG: clubedelphiphp.blogspot.com

Guinther Pauli
guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET, MCTS, MCPD
Delphi Certified: 6, 7, 2005, 2006, Web, Kylix

ClubeDelphi PLUS

DevMedia group

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

Assinatura

ClubeDelphi PLUS

Mais conteúdo Delphi por menos

Informativo ClubeDelphi

Portal ClubeDelphi

www.clubedelphi.net/portal

+600 vídeo aulas e 7 cursos online

Brinde na web desta edição

3 Vídeos

ASP.NET 2.0 - Treview
http://www.devmedia.com.br/articles/viewcomp.asp?comp=9555

RAD 2007, SQL Server Express e ASP.NET 2.0
http://www.devmedia.com.br/articles/viewcomp.asp?comp=9250

Novidades da nova versão do Delphi For PHP 2.0 - Parte I
http://www.devmedia.com.br/articles/viewcomp.asp?comp=9552

Para acessar os vídeos utilize os seguintes dados: Login: DVM.CD Senha: (multitier>

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Edições Anteriores da ClubeDelphi

Você pode comprar todas as edições anteriores da ClubeDelphi através do site DevMedia!

Para isso basta acessar *https://seguro.devmedia.com.br/edicoes_anteriores.asp*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link: *www.devmedia.com.br/clubedelphi/feedback*

Para votar, você vai precisar do código e senha de banca desta edição, que é:

Login: **DVM.CD** Senha: **(multitier>**

Ask The Expert

Perguntas e Respostas

Dúvidas respondidas por **Adriano Santos**
(envie as suas para falecom@adrianosantos.pro.br)

Listando estações de trabalho na rede

Muita gente me pergunta como fazer para listar estações da rede. Bem, o processo é um pouco complexo, mas vamos lá. Crie uma nova aplicação no Delphi e insira um *ListBox*("lstEstacoes") e um botão. No código do botão digite o código da **Listagem 1**.

Em seguida crie uma procedure, duas functions e duas variáveis na sessão *public* conforme segue:

```
Plataforma: string;
ListaUsuarios: TStringList;
procedure ListaAmbienteRede;
function GetIP(AEstacao: string): string;
function VersaoWindows: string;
```

Pressione *Ctrl* + *Shift* + *C* para que o Delphi crie o cabeçalho das funções. Em seguida digite o código de cada função conforme a **Listagem 2** observando o código de cada *function* e *procedure*.

Nos eventos *OnCreate* e *OnDestroy* do formulário, crie e destrua a variável *TStringList ListaUsuarios*, conforme a **Listagem 3**. O exemplo em funcionamento você pode visualizar na **Figura 1**. ●

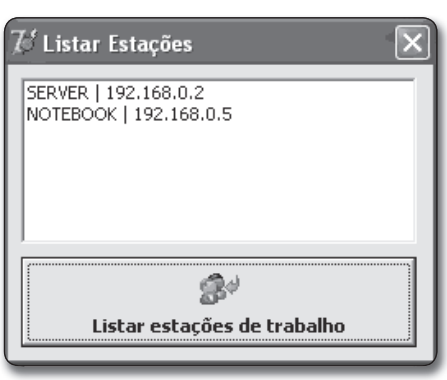


Figura 1. Sistema rodando

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Listagem 1. Código do botão Listar Estações

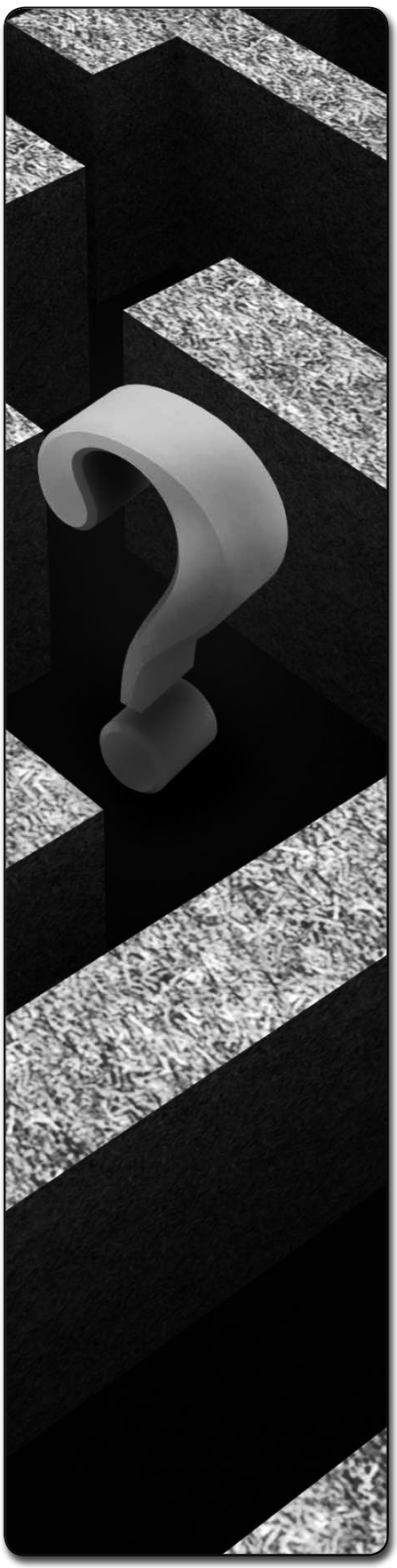
```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
  I: shortint;
  Nome: string;
  IP: string;
begin
  ListaUsuarios.Clear;
  lstEstacoes.Items.Clear;
  Plataforma := VersaoWindows;
  ListaAmbienteRede;
  lstEstacoes.Items.Clear;
  for I := 0 to ListaUsuarios.Count - 1 do
  begin
    Nome := ListaUsuarios[I];
    try
      IP := GetIP(ListaUsuarios[I]);
    except
      IP := ' ';
    end;
    lstEstacoes.Items.Add(Nome + ' | ' + IP);
    Update;
  end;
end;
```

Listagem 2. Códigos dos métodos localizar as estações

```
procedure TForm1.ListaAmbienteRede;
  procedure Enumera(Res: PnetResource);
  var
    Hnd: THandle;
    NumeroEntradas: DWord;
    Buffer: array[1..255] of TNetResource;
    LongBuffer: DWord;
    N: Integer;
    S: string;
  begin
    LongBuffer := SizeOf(Buffer);
    if WNetOpenEnum(RESOURCE_GLOBALNET, RESOURCETYPE_ANY, 0, Res, Hnd) = No_ERROR then
    begin
      NumeroEntradas := 255;
      WNetEnumResource(Hnd, NumeroEntradas, @Buffer[1], LongBuffer);
      for N := 1 to NumeroEntradas do
      begin
        S := string(Buffer[N].lpRemoteName);
        if Plataforma = 'xp' then
          if (Buffer[1].dwType = 0) and (Copy(S, 1, 2) = '\\') then
            ListaUsuarios.Add(Copy(S, 3, Length(S) - 2));
        if Plataforma = '9x' then
          if (Buffer[1].dwType = 3) and (Copy(S, 1, 2) = '\\') then
            ListaUsuarios.Add(Copy(S, 3, Length(S) - 2));
        if (Buffer[N].dwUsage and RESOURCEUSAGE_CONTAINER) = RESOURCEUSAGE_CONTAINER then
          Enumera(@Buffer[N]);
        Update;
      end;
    end;
  end;
begin
  Enumera(nil);
end;
function TForm1.GetIP(AEstacao: string): string;
var
  WSAData: TWSAData;
  HostEnt: PHostEnt;
begin
  WSAStartup(2, WSAData);
  HostEnt := GetHostByName(PChar(AEstacao));
  with HostEnt^ do
    Result := Format('%d.%d.%d.%d', [Byte(h_addr^[0]), Byte(h_addr^[1]),
      Byte(h_addr^[2]), Byte(h_addr^[3])]);
  WSACleanup;
end;
function TForm1.VersaoWindows: string;
var
  PlatformId, CSDVersion: string;
begin
  CSDVersion := '';
  case Win32Platform of
    VER_PLATFORM_WIN32_WINDOWS:
      begin
        if Win32MajorVersion = 4 then case Win32MinorVersion of
            0: PlatformId := '9x';
            10: PlatformId := '9x';
            90: PlatformId := 'ME';
          end
        else
          PlatformId := '9x';
        Result := '9x';
      end;
    VER_PLATFORM_WIN32_NT:
      begin
        if Length(Win32CSDVersion) > 0 then CSDVersion := Win32CSDVersion;
        if Win32MajorVersion <= 4 then PlatformId := 'NT'
        else if Win32MajorVersion = 5 then
          case Win32MinorVersion of
            0: PlatformId := '2000';
            1: PlatformId := 'XP';
            2: PlatformId := '2003';
          else
            PlatformId := '?';
          end
        else
          PlatformId := '?';
        Result := 'xp';
      end;
  end;
end;
```

Listagem 3. Criação de destruição de TStringList

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ListaUsuarios := TStringList.Create;
end;
procedure TForm1.FormDestroy(Sender: TObject);
begin
  ListaUsuarios.Free;
end;
```



Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

POO e Multicamadas

Boas práticas de desenvolvimento com ASP.NET 2.0

Nesse artigo veremos

- Criação de uma aplicação modelo com ASP.NET 2.0, AJAX e POO;
- Divisão da aplicação em camadas;
- Criação de camadas DAL, BLL e de uma DTO;
- Transferência de objetos entre camadas;
- Separação da lógica de negócio e acesso a dados da interface da aplicação.

Qual a finalidade

- Criar aplicações mais robustas, fáceis de manter, ao mesmo tempo que utiliza boas práticas de desenvolvimento.

Quais situações utilizam esses recursos?

- Na opinião do autor, todos os WebSites deveriam ser desenvolvidos dessa forma;

Na arquitetura de uma aplicação *ASP.NET* tradicional, normalmente utilizamos um banco de dados relacional e criamos uma interface com controles *data-bound*. Com os novos controles de acesso a dados introduzidos pelo *ASP.NET 2.0*, podemos utilizar um *SqlDataSource* ou *dbxDataSource*, por exemplo, para conectar a um banco de dados e manipular informações em tela.

Mas será que essa é a melhor forma de desenvolver aplicações *ASP.NET*? Será que é a maneira correta? Não é estranho ter funcionalidades de acesso a dados combinadas com controles de *UI*? E a *POO*, como tirar do papel e colocar em prática? É muito comum um desenvolvedor, pela pressão em finalizar logo um projeto ou mesmo por falta de conhecimento, sair colocando código *SQL* no próprio *Web Form*. E essa com certeza não é uma boa prática.

O objetivo deste artigo é ensinar você leitor a aplicar boas práticas de desenvolvimento de aplicações *ASP.NET*. Você aprenderá, através de um exemplo prático passo a passo, como construir uma solução Web utilizando boas práticas de desenvolvimento.

O que vamos desenvolver é uma aplicação distribuída em camadas, utilizando o melhor do que a *POO* (Programação

Guinther Pauli
guinther@devmedia.com.br
é Bacharel em Sistemas de Informação, Microsoft Certified: MCP, MCAD, MCSD.NET, MCTS, MCPD, Borland Certified: Delphi 6, 7, 2005, 2006, Web e Kylix. Editor Geral das Revistas .net Magazine e ClubeDelphi. Desenvolve em ASP.NET há 8 anos.

Orientada a Objetos) tem a oferecer. Podemos dizer que nossa solução será praticamente 100% Orientada a Objetos.

A metodologia que utilizarei é simples, porém interessante. Iniciaremos criando uma aplicação da forma tradicional, com código de acesso a dados mesclado com o código de interface. A seguir, vamos começar a separar a aplicação em partes (camadas), incluindo código e funcionalidade em cada uma. O porquê de dividir em camadas é explicado a seguir.

Antes de iniciar, vou apresentar o nosso "mapa". A **Figura 1** representa a solução que vamos desenvolver. É importantíssimo que você entenda cada parte da aplicação multicamadas e as siglas utilizadas, pois elas serão referenciadas durante todo este tutorial.

Durante o desenvolvimento da solução, sempre que tiver uma dúvida em relação a qual camada estamos implementando, determinada funcionalidade, consulte essa figura.

Explicando as camadas:

• Banco de Dados (*DB*): será um banco no *SQL Server 2005 Express*;

• *Data Access Layer* (*DAL*): é a camada que cuida e centraliza o acesso a dados. Aqui ficam os códigos para inserir e atualizar dados nas tabelas do banco e onde são utilizadas as classes do *ADO.NET*, ou seja, é a camada encarregada da persistência dos dados;

• *Business Logic Layer* (*BLL*): camada de regra de negócio, funciona como "ponte" entre a *DAL* e *DB*. Contém regras de negócio da aplicação, validações a nível de objeto etc.

• *Presentation* ou *Interface*: a aplicação *ASP.NET* propriamente dita. Note que podemos utilizar outro tipo de interface, como *Windows Forms* ou um dispositivo móvel (com o auxílio de um *Web Service*).

Nota do DevMan

Podemos dizer que a DAL faz o papel do *DataModule* em aplicações tradicionais em Delphi, pois encapsulam o acesso a dados em um camada distinta. No entanto, emular o uso de *DataModules* no *ASP.NET* de outra forma, como criando uma classe herdada de *Component*, é uma má prática de programação Web e pode degradar a performance da sua solução.

Por que desenvolver dessa forma (vantagens)

Muitos podem estar se perguntando: Por que eu devo desenvolver dessa forma e não da maneira como estou acostumado? Quais as reais vantagens que eu ganho? Vamos enumerar os principais benefícios:

• A camada de apresentação contém apenas código de *UI*, ela não acessa o banco de dados diretamente, não tem conhecimento sobre tabelas, esquemas, qual o tipo do banco de dados, apenas cuida de apresentar os dados para o usuário e manipulá-los. Com isso, podemos dizer que a aplicação cliente é *thin-client* (cliente leve). Isso nos dá um nível elevado de *abstração*;

• Independência de tipo de aplicação cliente: como o acesso a dados é centralizado, podemos compartilhar esse código de acesso entre vários tipos de aplicações cliente, como *Web, Desktop e Mobile;*

• Migração facilitada: Como estamos separando a solução em camadas, uma possível migração fica muito simples, já que tudo está separado. Por exemplo, se quisermos migrar de *SQL Server* para *Oracle*, apenas uma camada (a *DAL*) precisaria sofrer alguns ajustes, as outras camadas não enxergam esse acesso;

• Reaproveitamento de código: o código de acesso a dados (e *BLL*), por exemplo, não é replicado caso tenhamos vários tipos de aplicativos cliente ou caso tenhamos múltiplas interfaces fazendo uso dos mesmos objetos;

• Fácil substituição: para explicar a *POO*, muito recorre-se à vida real para fazer comparativos, vou fazer o mesmo aqui. Imagine um computador, ele é formado por vários objetos, como se fossem "camadas" que se comunicam. Se uma peça queimar ou você quiser fazer um *upgrade*, precisará trocar somente aquele objeto, e não o computador todo. O mesmo acontece em nossa solução. Se você desejar implementar um mecanismo mais robusto de acesso a dados, basta criar uma nova camada *DAL* e substituir na arquitetura, as demais partes permanecem inalteradas;

• Manutenção facilitada: como os aplicativos estão em camadas, se houver um erro ou for necessário fazer uma manutenção / correção, normalmente apenas uma camada é afetada, o que também facilita o *deploy*;

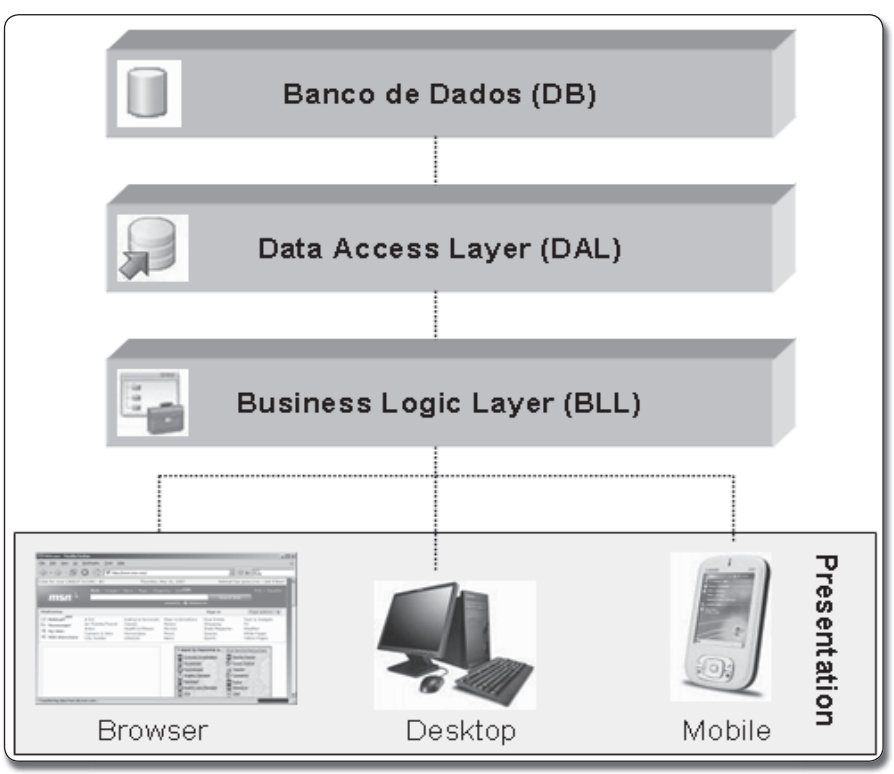


Figura 1. Arquitetura da solução multicamadas

• E finalmente, estaremos usando o melhor da *POO*. Eu fico imaginando como não era trabalhoso e árduo trabalhar com o *ASP* clássico sem esses recursos (o *include* era o melhor recurso para "herança"!). Em nossa solução vamos usar e desfrutar de recursos da *POO* pura, como classes, herança, encapsulamento e muito mais, além de um novo e poderoso recurso do *ASP.NET 2.0*: o *ObjectDataSource*.

Explicados os fundamentos, arquitetura e vantagens da solução proposta, vamos para a prática.

Nota do DevMan

O ObjectDataSource faz parte da família dos Data Sources, porém, ao invés de se conectar diretamente a uma base de dados, como faz o SqlDataSource, ele se conecta a uma classe de negócio.

Links

Neste artigo, utilizaremos o *SQL Server 2000 Express*, free, comprovadamente o MELHOR banco para se trabalhar com aplicações *ASP.NET*. Para gerenciar o banco de dados, utilizaremos uma ferramenta chamada *SQL Server Management Studio Express* (semelhante ao que é o *IBExpert* para o *Firebird*). Ambos os *downloads* estão listados abaixo, a instalação é muito simples e não detalharei aqui.

SQL Server 2005 Express

http://www.microsoft.com/express/download/

SQL Server 2005 Express e Management Studio Express

http://msdn.microsoft.com/vstudio/express/sql

Criando o banco de dados

Nosso aplicativo vai gerenciar edições de uma revista, nesse caso a ClubeDelphi. Caso queira, você poderá utilizar um banco de dados pronto, como o *Northwind*.

Utilizando o *SQL Server Management Studio Express*, após efetuar o *Logon*, crie um novo banco (dando um clique de direita em *Databases*) chamado "EDITORA". Nele, crie uma nova tabela chamada "Revistas", usando o esquema da **Figura 2.** Para isso, dê um clique de direita sobre EDITORA>*Tables* e escolha *New Table*. Para definir o campo NUM_EDICAO como chave, basta selecioná-lo e clicar sobre o ícone de chave.

Nota: Para fins didáticos, simplificamos ao máximo a estrutura da tabela, não normalizando alguns campos que precisariam ser normalizados. Além disso, vamos trabalhar com uma única tabela em nosso banco de dados, facilitando o exemplo e para que possamos nos concentrar nos aspectos relativos à *POO* e multicamadas.

Após criar a tabela (salve-a com o nome de REVISTAS), insira alguns registros usando o próprio gerenciador (opção *Open Table*). Cadastre algumas revistas (**Figura 3**), no meu caso cadastrei as últimas edições da ClubeDelphi. O campo *Capa* indica qual foi a matéria de capa e o campo *Nivel* indica a média do nível da revista, entre avançado e iniciante, onde 3 indica iniciante, 2 intermediário e 1 avançado.

Column Name	Data Type	Allow Nulls
NUM_EDICAO	int	☐
CAPA	varchar(250)	☐
NIVEL	decimal(18, 2)	☐
		☐

Figura 2. Tabela de Revistas

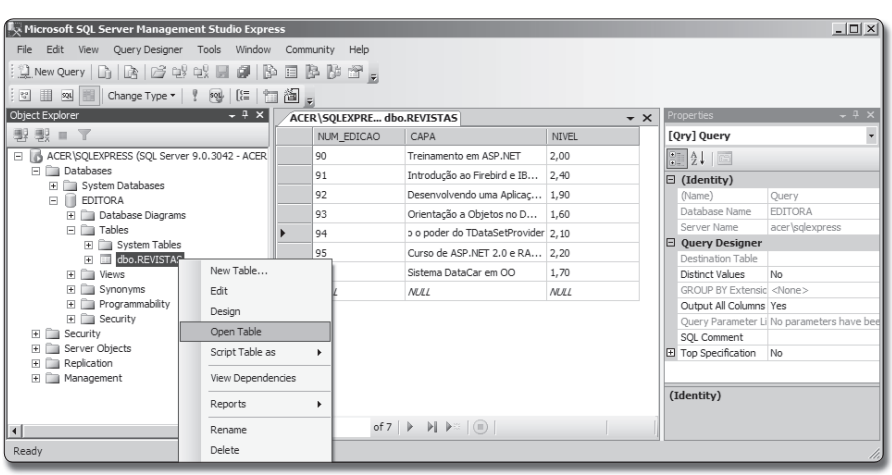


Figura 3. Inserindo registros para teste

Construindo a aplicação

No *RAD Studio 2007*, clique em *File>New>Other* e escolhe a opção *AJAX-Enabled ASP.NET Web Application* (**Figura 4**)**.** Dê o nome de *Multitier* para a solução e salve-a na pasta *c:*, escolhendo o servidor Web (*Server*) como *Cassini Web Server*). O *Cassini* irá nos ajudar pois não precisaremos usar o *IIS*, ele serve como um servidor local para aplicação Web (**Figura 5**).

Conforme comentei, vamos inicialmente construir uma aplicação padrão, semelhante às aplicações que encontramos, sem separação em camadas. Em um primeiro momento, vamos fazer tudo na interface cliente, depois vamos começar a dividir em camadas.

Criando a conexão

A interface é simples, temos um *GridView* e um *DetailsView* ligados a um *SqlDataSource*, que por sua vez está conectado ao banco *Editora* e recupera todos os registros da tabela *Revistas*. Para configurar o *SqlDataSource*, basta colocar o componente (paleta *Web Data*) e escolher a opção *Configure Data Source* nas *Tasks*. Nota: *O SqlDataSource* deve ficar fora do *UpdatePanel*.

Se não aparecer o nome do banco, clique em *New Connection* e configure conforme mostrado nas **Figuras 6** e **7**. Observe que devemos escolher o provider para *SQL Server* e na tela seguinte digitar o nome do servidor,

que consiste do NOME_DA_MAQUINA\SQLEXPRESS. Depois, basta selecionar o banco de dados EDITORA e clicar em Ok.

Clique em *Next*, *Next* e na tela seguinte selecione todos os campos da tabela REVISTAS (**Figura 8).** Clique ainda no botão *Advanced* e marque a opção para habilitar os *updates*. Clique em *Next* e *Finish*.

Criando a interface

Ambos *DetailsView* e *GridView* (paleta *Web Data*) estão dentro de um *UpdatePanel* do *ASP.NET AJAX* (coloque um a partir da paleta de componentes), o que permitirá operações assíncronas e sem *postbacks* ao servidor, como a edição e atualização de dados, graças ao *AJAX*. Lembre de ligá-los ao *SqlDataSource* através das *Tasks* na opção *Choose Data Source*. Se preferir, ainda nas *Tasks*, escolha *Auto Format* para formatar os controles.

O *DetailsView* servirá apenas para a inserção, então defina sua propriedade *DefaultMode* para *Insert* e *AutoGenerateInsertButton* para *True*. O *GridView* fará a edição, então habilite a opção *Enable Editing* nas *tasks* do controle. Habilite também *Enable Paging* e *Enable Sorting*.

Executando a aplicação, sua tela deve estar semelhante à **Figura 9**. Inclua alguns registros, pagine o *GridView* (para melhor visualização, define seu *PageSize* para 3), ordene, edite, e observe que tudo é feito sem *postbacks*, graças ao *ASP.NET AJAX*. Com isso sua aplicação Web fica muito mais rápida, muito mais interativa com o usuário (um dos princípios da Web 2.0).

Data Acess Layer – DAL (Camada de Acesso a Dados)

Nosso primeiro passo vai ser retirar o código de acesso a dados da interface. Nossa camada de acesso a dados (*DAL*) será composta de apenas uma classe, chamada *Revista*, que basicamente contém os métodos para manipular os dados, conhecidos também como métodos CRUD (*Create, Read, Update* e *Delete*).

Você pode incluir métodos personalizados para, por exemplo, retornar todas as revistas que tiveram nível avançado. A nossa camada *DAL* será criada em uma *unit* a parte, para indicar a separação em camadas.

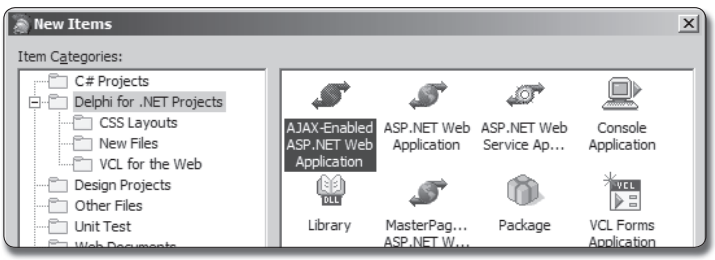


Figura 4. Criando uma aplicação com AJAX

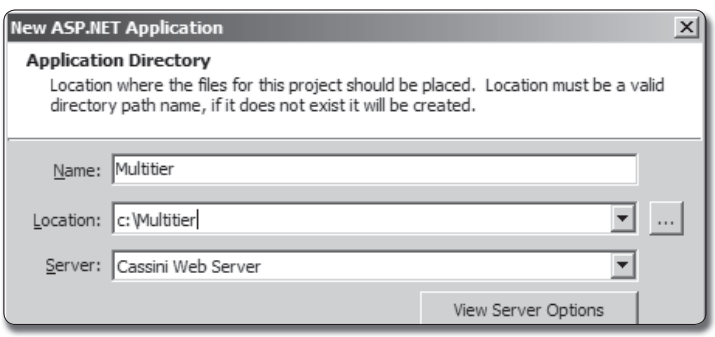


Figura 5. Escolhendo o Cassini Web Server

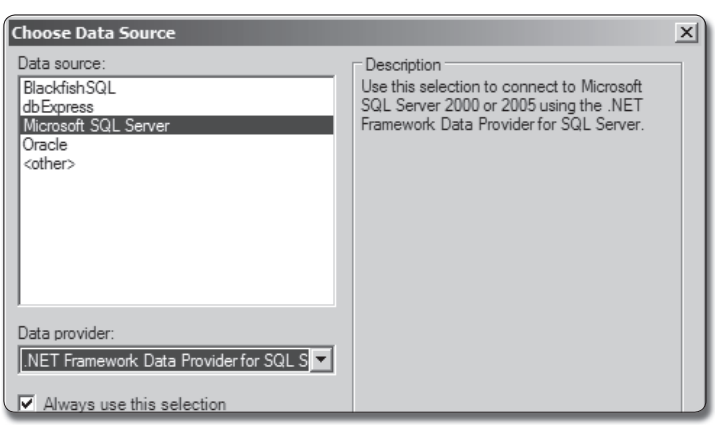


Figura 6. Escolhendo a conexão

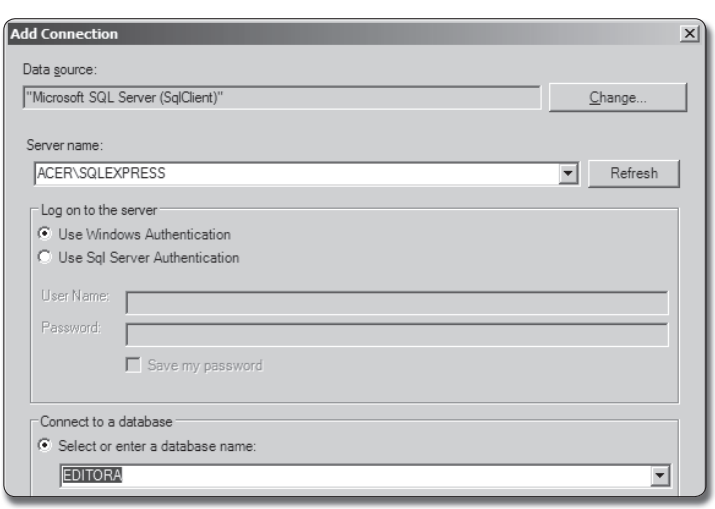


Figura 7. Localizando a máquina e banco de dados

Listagem 1. Código da classe Revista da DAL

```
unit DAL;
interface
uses System.Data,
     System.Data.SqlClient;
type
  Revista = class
  public
    function Select(): DataSet;
    procedure Update(NUM_EDICAO: integer; CAPA: System.string; NIVEL: double);
    procedure Insert(NUM_EDICAO: integer; CAPA: System.string; NIVEL: double);
    procedure Delete(NUM_EDICAO: integer);
  end;
implementation
const
    ConStr : String  = 'Data Source=ACER\'+
      'SQLEXPRESS;Initial Catalog=EDITORA;Integrated'+
      ' Security=True';
function Revista.Select(): DataSet;
var
  Con: SqlConnection;
  Cmd: SqlCommand;
  da: SqlDataAdapter;
  SQL: string;
  ds: DataSet;
begin
  Con := SqlConnection.Create(ConStr);
  SQL := 'select * from REVISTAS';
  Cmd := SqlCommand.Create(SQL, Con);
  da := SqlDataAdapter.Create(Cmd);
  ds := DataSet.Create();
  da.Fill(ds);
  result := ds;
 end;
procedure Revista.Update(NUM_EDICAO: integer; CAPA: System.string; NIVEL: double);
var
  Con: SqlConnection;
  Cmd: SqlCommand;
  SQL: string;
begin
  Con := SqlConnection.Create(ConStr);
  SQL := 'update REVISTAS set ' +
    'CAPA=@CAPA,NIVEL=@NIVEL ' +
    'where NUM_EDICAO=@NUM_EDICAO_Original';
  Cmd := SqlCommand.Create(SQL, Con);
  Cmd.Parameters.AddWithValue('@CAPA',CAPA);
  Cmd.Parameters.AddWithValue('@NIVEL',&object(NIVEL));
  Cmd.Parameters.AddWithValue('@NUM_EDICAO_Original',&object(NUM_EDICAO));
  Con.Open();
  try
    Cmd.ExecuteNonQuery();
  finally
    Con.Close();
  end;
end;
procedure Revista.Insert(NUM_EDICAO: integer; CAPA: System.string; NIVEL: double);
var
  Con: SqlConnection;
  Cmd: SqlCommand;
  SQL: string;
begin
  con := SqlConnection.Create(ConStr);
  SQL := 'insert into REVISTAS' + 'values(@CAPA,@NIVEL,@NUM_EDICAO)';
  Cmd := SqlCommand.Create(SQL, Con);
  Cmd.Parameters.AddWithValue('@CAPA',CAPA);
  Cmd.Parameters.AddWithValue('@NIVEL',&object(NIVEL));
  Cmd.Parameters.AddWithValue('@NUM_EDICAO',&object(NUM_EDICAO));
  Con.Open();
  try
    Cmd.ExecuteNonQuery();
  finally
    Con.Close();
  end;
end;
procedure Revista.Delete(NUM_EDICAO: integer);
var
  Con: SqlConnection;
  Cmd: SqlCommand;
  SQL: string;
begin
  Con := SqlConnection.Create(ConStr);
  SQL := 'delete REVISTAS' + 'where NUM_EDICAO=@NUM_EDICAO)';
  Cmd := SqlCommand.Create(SQL, Con);
  Cmd.Parameters.AddWithValue('@NUM_EDICAO',&object(NUM_EDICAO));
  Con.Open();
  try
    Cmd.ExecuteNonQuery();
  finally
    Con.Close();
  end;
end;
end.
```

No *Project Manager*, dê um clique de direita no nome do projeto e escolher *Add New>Folder*, dando o nome de *App_Code* para a pasta. Atenção, use exatamente este nome pois esta é uma pasta especial do *ASP.NET*. Clique de direita sobre essa pasta e escolha *Add New>Other>Delphi for .NET Projects>New Files>Class*. Salve a *unit* como *DAL.pas* e codifique a nova classe conforme mostrado na **Listagem 1**.

Aqui estamos basicamente utilizando classes do *ADO.NET* para comunicação com o banco de dados. O *Select* utiliza um *SqlDataAdapter* parar preencher um *DataSet* e devolve para a camada que chamou a *DAL*. O *Update, Insert* e *Delete* utilizam *SqlCommands* parar executar comandos *DML* no banco. Note que eles recebem os valores dos parâmetros das instruções *SQL* diretamente nos parâmetros da assinatura do método, como, por exemplo, NUM_EDICAO, CAPA e NIVEL.

Lembre-se de adaptar a *string* de conexão para refletir sua configuração. Uma opção seria armazenar essa *string* em um arquivo de configuração.

Agora precisamos fazer com que a camada de interface (*Web Site*) reconheça a *DAL*. Dê um *Build* no projeto e vá até a aplicação de interface. Na interface principal do *Web Site*, coloque um *ObjectDataSource* (*ODS*) e retire o *SqlDataSource* (o componente deve ficar dentro do *UpdatePanel*). Configure o *DataSourceID* dos controles de tela para apontarem para o *ODS*. Nas *tasks* do novo controle ative a opção *Configure Data Source* para abrir um assistente de configuração.

O *ObjectDataSource* também é um tipo *IDataSource*, assim como o *SqlDataSource*. É utilizado para criar aplicativos em camadas, já que pode reconhecer um objeto de negócio como fonte de dados e chamar métodos para obter e atualizar dados. Nesta primeira etapa, o objeto de negócio será a classe *Revista* da *DAL*. Diferente do *SqlDataSource*, que armazena instruções *SQL* junto com a interface, o *ODS* apenas referencia a classe de negócio.

Então, em *Choose a Business Object* (nas *Tasks* do *ODS*) escolha *DAL.Revista* **(Figura 10).** Na próxima tela em *Select* escolha o método de mesmo nome (**Figura 11)** e repita o mesmo passo para os demais métodos. Clique a seguir em

Finish. Execute a aplicação e faça testes de seleção, inserção, *update* etc.

Nesse momento, depois da exclusão do *SqlDataSource*, está tudo funcionando perfeitamente. Nossa aplicação já é multicamadas (e veja que a interface não muda aparentemente em nada). Temos o banco de dados, a camada *DAL* e a interface. Muitos desenvolvedores já poderiam parar por aqui e optar por essa arquitetura, que já é bastante flexível. Mais adiante vamos dar um passo além e criar mais uma camada na solução, a *BLL*.

Note que nossa camada *DAL* devolve *DataSets* para a camada de interface, mas poderia trocar outros tipos de dados, como *IDataReaders*, documentos *XML* ou classes customizadas, minha opção preferida (veja mais adiante).

Nota: Uma outra boa prática seria usar *Data Access Application Blocks* nessa camada.

Stored Procedures

Já que estamos falando em boas práticas não poderíamos deixar de falar em *Stored Procedures*. Primeiro, o uso de *SPs* adiciona performance. Além disso, você mantém todas as instruções *SQL* no próprio *BD* e esconde da aplicação detalhes sobre estruturas de tabelas, criando mais um nível de isolamento em nossa solução.

Usando o *Management Studio Express*, abra o banco criado e no item *Programmability* dê um clique de direita sobre o item *Stored Procedures*. Usando a opção *New Procedure*, crie as *procedures* mostradas na **Listagem 2.** A seguir, clique no botão *Execute*.

Modifique a camada *DAL* para que faça uso das *Stored Procedures*, bastando para isso configurar os *SqlCommands*. Por exemplo, o método para seleção fica:

```
Cmd := SqlCommand.Create('SelectRevistas',
Con);
Cmd.CommandType := CommandType.
StoredProcedure;
```

Nota

No *ASP.NET* não existe o conceito de *DataModules*. O que deve ser feito, nesse caso, para isolar o código de acesso a dados, é criar uma camada separada. Essa é a nossa *DAL*. Ela funciona de forma semelhante a um *DataModule*, exceto pelo fato que você deve codificar tudo manualmente.

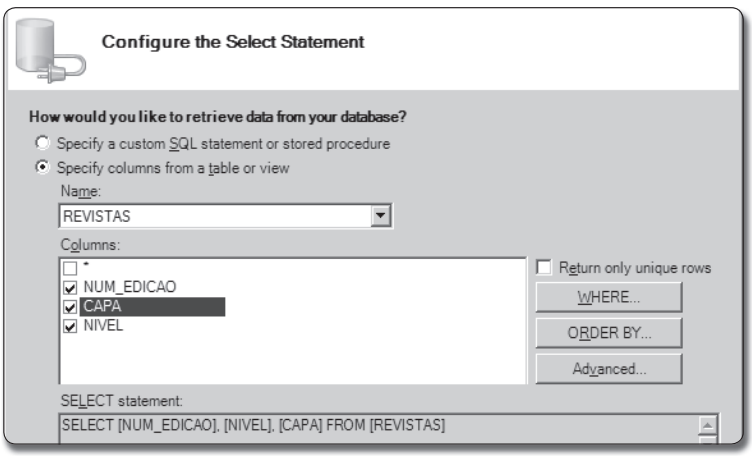


Figura 8. Selecionando campos da tabela Revistas

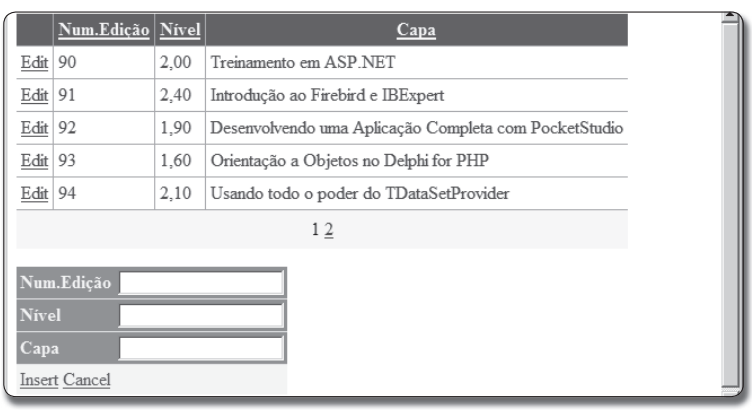


Figura 9. Interface da aplicação usando AJAX

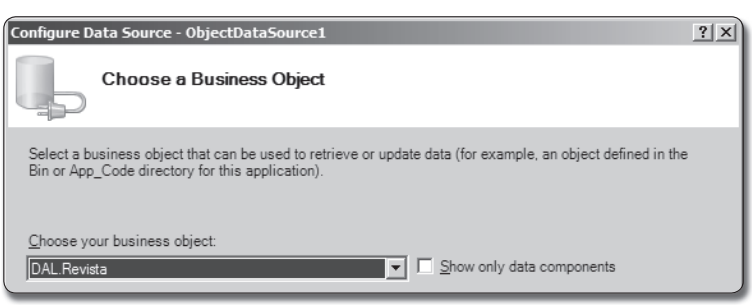


Figura 10. Escolhendo a entidade de negócio

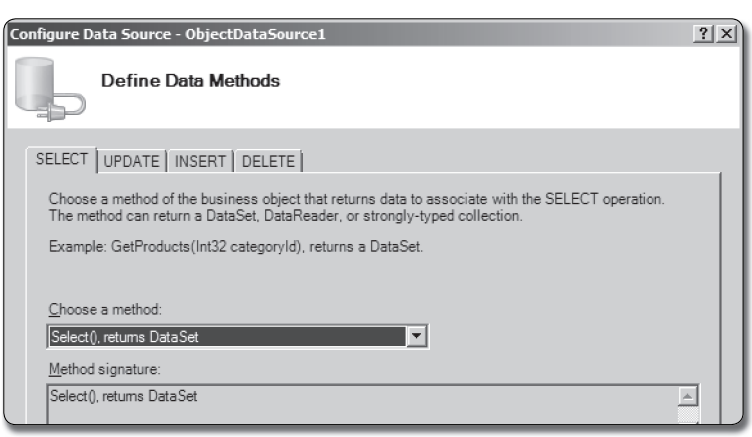


Figura 11. Escolhendo o método de seleção

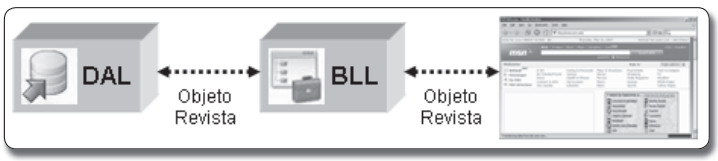


Figura 12. Troca de objetos entre camadas

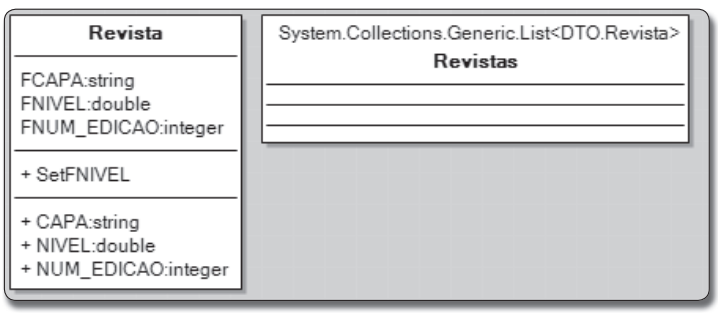


Figura 13. Modelando classes (entidades)

Listagem 2. Código das Stored Procedures

```
-- SELECT
create procedure SelectRevistas
as
begin
select * from REVISTAS
end
GO
-- UPDATE
create procedure UpdateRevista
  @NUM_EDICAO int,
  @CAPA varchar(250),
  @NIVEL decimal(18,2)
as
begin
update REVISTAS set CAPA = @CAPA, NIVEL = @NIVEL
    where NUM_EDICAO = @NUM_EDICAO
end
GO
-- INSERT
create procedure InsertRevista
  @NUM_EDICAO int,
  @CAPA varchar(250),
  @NIVEL decimal(18,2)
as
begin
insert into REVISTAS values (@NUM_EDICAO,@CAPA,@NIVEL)
end
GO
-- DELETE
create procedure DeleteRevista
  @NUM_EDICAO int
as
begin
delete REVISTAS where NUM_EDICAO = @NUM_EDICAO
end
GO
```

Listagem 3. Criando as entidades da aplicação

```
unit DTO;
interface
uses System.Collections.Generic;
type
  Revista = class
  private
    FNUM_EDICAO: integer;
    FNIVEL: double;
    FCAPA: string;
  published
  public
    property NUM_EDICAO: integer read FNUM_EDICAO write FNUM_EDICAO;
    property CAPA: string read FCAPA write FCAPA;
    property NIVEL: double read FNIVEL write FNIVEL;
  end;
  Revistas = class (List<Revista>) //generics
  private
  end;
implementation
end.
```

Observe que no lugar do comando *SQL* agora passamos o nome da *SP* a ser utilizada. A seguir, configuramos o *CommandType* do *SqlCommand* para indicar que vamos executar uma *SP*. A variável *SQL* e sua *string* foram removidas pois não é mais necessário armazenar comandos *SQL* na *DAL*.

Faça o mesmo procedimento para os demais métodos e certifique-se que o nome de cada parâmetro que esteja passando na collection *Parameters* corresponda ao nome do parâmetro que está na *SP*.

Execute e teste a aplicação para uso com *SPs*. Veja que nada muda na interface, ela continua funcionando da mesma forma que no início do artigo. Os reais ganhos são internos à arquitetura, com performance, isolamento e abstração.

Criando Entidades de Negócio

A seguir, vamos criar mais uma camada em nossa solução, a *BLL*. Nesse caso, informações sobre revistas precisam ser trafegadas de uma camada para outra da aplicação. Em nosso exemplo, até agora, estamos usando *DataSets* para devolver dados para a interface e métodos com parâmetros usando tipos escalares (*int*, *string*, *double*) para atualizar dados.

O que vamos fazer agora é definir uma classe que representará a entidade *Revista*, ou seja, a representação *OO* de uma tabela do modelo relacional (processo conhecido como mapeamento O/R- objeto/relacional). Essa entidade será utilizada para comunicar as diferentes camadas da aplicação.

Aqui optamos por fazer um mapeamento 1-1 (uma tabela representa uma classe), mas nada impede de você manipular mais de uma tabela em uma única classe. Por exemplo, nossa classe *Revista* poderia ter métodos para manipular ambas as tabelas de *Revistas* e *Artigos* (caso houvesse). O modelo de classes não precisa ser um espelho do modelo relacional.

A camada de interface não vai mais receber um *DataSet* para exibir os dados, mas uma coleção de objetos do tipo *Revista*. Quando for atualizar os dados, não chamará um método passando os valores a serem atualizados, mas um objeto contendo em seus atributos os valores a serem persistidos no *BD*. A **Figura 12** representa a troca de objetos customizados entre as camadas da aplicação.

Para criar a nova classe, vamos definir uma nova *unit*, pois ela precisará ser acessada e compartilhada por todas as camadas. Clique de direita sobre a pasta *App_Code* e crie uma nova classe semelhante ao que foi feito anteriormente. Dê o nome de *DTO_Revista* para a *unit* e implemente-a conforme a **Listagem 3.** Obs.: a sigla *DTO* vem de *Data Transfer Objects*, ou objetos para transferência de dados.

A classe mapeia fielmente a tabela de revistas, tendo uma propriedade para cada campo da tabela. Para armazenar os valores, são utilizados campos (*fields*) privados que são internos à classe.

Note que logo abaixo da classe *Revista* temos a classe *Revistas*. Aqui usamos um novo recurso do *Delphi* para o *.NET 2.0*, chamado *Generics*, para criar uma lista fortemente tipada de objetos *Revista*. Esse tipo será usado para representar uma coleção de objetos, normalmente retornada por um método *Select*.

Caso queira usar o *designer* de diagrama de classes do *RAD Studio*, basta acioná-lo a partir do *Project Manager* (*Model View*) e desenhar as classes visualmente, conforme mostra a **Figura 13.** Você pode criar os *fields* e as propriedades associadas.

Business Logic Layer – BLL (Camada Lógica de Negócio)

A função dessa camada é abstrair a camada *DAL* da camada de interface, funcionando como um "meio de campo". Nessa camada você pode implementar regras de negócio específicas para tratar determinados dados, como validações, realizar cálculos e manipulação de dados antes de serem enviados à interface ou serem atualizados pela *DAL* (como definição de valores default). Pode também filtrar os dados a serem exibidos de acordo com as credenciais de um usuário, ou mesmo inibir determinadas operações. Enfim, toda regra relacionada a um objeto.

Ela também pode funcionar como uma ponte caso você decida, por exemplo, colocar a *DAL* em um *Web Service*, e permitir acesso à solução a partir de dispositivos móveis.

Para implementar a camada, na pasta *App_Code*, crie uma nova *unit* chamada *BLL.pas*. Implemente a classe da *BLL* como mostrado na **Listagem 4.**

> **Nota**: Veja que todas as camadas da solução definem uma classe chamada *Revista*. Resolvi dar o mesmo nome, pois o *namespace* (nome da *unit*) se encarrega de identificar a classe dentro do seu contexto e designar sua função. Por exemplo, *DAL.Revista* é a classe que persiste os dados, *DTO.Revista* é a entidade do mapeamento O/R, *BLL.Revista* é a classe de negócio.

Note que, por enquanto, tudo que a camada faz é receber/enviar dados da camada de apresentação e "passar a bola" para a *DAL*. Mais adiante veremos como personalizar essa camada com validações.

Adaptando a DAL para trabalhar com objetos fortemente tipados

Agora precisamos alterar a *DAL* para que os métodos da classe *Revista* rece-

Listagem 4. Classe de negócio da aplicação (BLL)

```
unit BLL;
interface
uses DTO, DAL;
type
  Revista = class
   public
    function Select(): DTO.Revistas;
    procedure Update(Rev: DTO.Revista);
    procedure Delete(Rev: DTO.Revista);
    procedure Insert (Rev: DTO.Revista);
  end;
implementation
procedure Revista.Insert(Rev: DTO.Revista);
var
  DAL_Rev: DAL.Revista;
begin
  DAL_Rev := DAL.Revista.create();
  DAL_Rev.Insert(Rev);
end;
function Revista.Select: DTO.Revistas;
var
  DAL_Rev: DAL.Revista;
begin
  DAL_Rev := DAL.Revista.create();
  result := DAL_Rev.Select();
end;
procedure Revista.Delete(Rev: DTO.Revista);
var
  DAL_Rev: DAL.Revista;
begin
  DAL_Rev := DAL.Revista.create();
  DAL_Rev.Delete(Rev);
end;
procedure Revista.Update(Rev: DTO.Revista);
var
  DAL_Rev: DAL.Revista;
begin
  DAL_Rev := DAL.Revista.create();
  DAL_Rev.Update(Rev);
end;
end.
```

Listagem 5. Select agora retorna uma lista de objetos

```
function Revista.Select: DTO.Revistas;
var
  Objs: DTO.Revistas;
  Con: SqlConnection;
  Cmd: SqlCommand;
  rd: SqlDataReader;
  Rev: DTO.Revista;
begin
  Objs := DTO.Revistas.Create();
  Con := SqlConnection.Create(ConStr);
  Cmd := SqlCommand.Create('SelectRevistas',Con);
  Cmd.CommandType := CommandType.StoredProcedure;
  Con.Open();
  rd := Cmd.ExecuteReader(CommandBehavior.CloseConnection);
  while (rd.Read()) do
  begin
    Rev := DTO.Revista.Create();
    Rev.Num_Edicao := Convert.ToInt32(rd[0]);
    Rev.Capa := rd[1].ToString();
    Rev.Nivel := Convert.ToDouble(rd[2]);
    Objs.Add(Rev);
  end;
  result := Objs;
  Con.Close();
end;
```

Listagem 6. Update agora recebe um objeto tipado como parâmetro

```
procedure Revista.Update(Rev: DTO.Revista);
var
  Con: SqlConnection;
  Cmd: SqlCommand;
  SQL: string;
begin
  Con := SqlConnection.Create(ConStr);
  SQL := 'UpdateRevista';
  Cmd := SqlCommand.Create(SQL, Con);
  Cmd.CommandType := CommandType.StoredProcedure; // new
  Cmd.Parameters.AddWithValue('@CAPA',Rev.CAPA);
  Cmd.Parameters.AddWithValue('@NIVEL',&object(Rev.NIVEL));
  Cmd.Parameters.AddWithValue('@NUM_EDICAO',&object(Rev.NUM_EDICAO));
  Con.Open();
  try
    Cmd.ExecuteNonQuery();
  finally
    Con.Close();
  end;
end;
```

Listagem 7. Validações a nível de objeto

```
procedure Revista.Insert(Rev: DTO.Revista);
var
  DAL_Rev: DAL.Revista;
begin
  if Rev.NIVEL > 2.5 then
     raise Exception.Create('Revista muito avançada');
  if Rev.NIVEL < 1.5 then
     raise Exception.Create('Revista muito introdutória');
  DAL_Rev := DAL.Revista.create();
  DAL_Rev.Insert(Rev);
end;
```

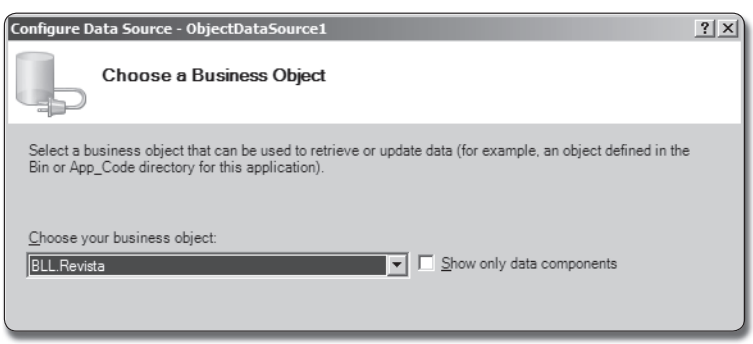


Figura 14. Informando ao ODS para usar a BLL como fonte de dados

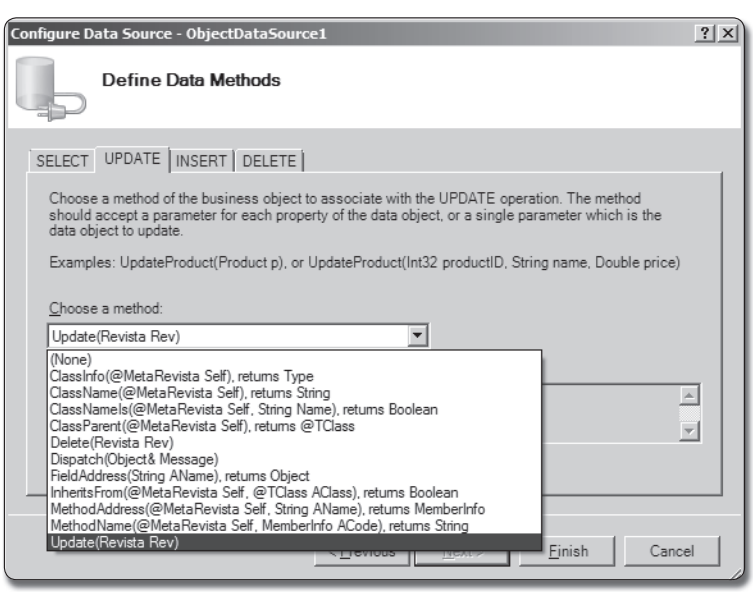


Figura 15. Definindo o método Update

bam e devolvam como parâmetro não mais tipos escalares, mas objetos. Por exemplo, o *Select* da *DAL* não retorna mais um *DataSet*, mas sim uma coleção de objetos fortemente tipados, nesse caso *Revistas*. A nova implementação dele é mostrada na **Listagem 5**.

Note que estamos usando um *DataReader* para ler os registros da tabela. Para cada registro encontrado, criamos um objeto do tipo *Revista* e adicionamos na coleção genérica que será devolvida como retorno do método. Note também que acessamos os campos do *DataReader* pelo índice, o que é mais rápido do que acessar pelo nome.

A **Listagem 6** mostra como fica o *Update*. Agora ele recebe por parâmetro não mais os valores de tipos escalares, mas um objeto do tipo *DTO.Revista*, nossa entidade de negócio. A outra modificação foi fazer os parâmetros obterem seus valores a partir desse objeto, o restante não muda. Utilize esse mesmo princípio para adaptar os métodos *Insert* e *Delete*.

Adaptando a interface para usar a BLL e tipos customizados

A interface da aplicação não se comunicará mais diretamente com a *DAL*, mas através da *BLL*, finalizando assim, nossa arquitetura exatamente como mostrada na **Figura 1**.

Além disso, ela não receberá mais dados em formato de *DataSet* e não chamará métodos da *DAL*. Toda a comunicação usará como "protocolo" o tipo customizado que definimos no *DTO*. O próximo passo é configurar o *ObjectDataSource*. No seu menu *Tasks*, escolha a opção *Configure Data Source*.

Na primeira tela que aparece, dentre os vários objetos de negócio que aparecem, escolha nossa classe *Revista* que está dentro da *BLL* (**Figura 14**), pois essa é a única camada que a interface deve interagir.

Configure agora os métodos *CRUD* utilizados pelo *ObjectDataSource*. Note por exemplo que para o *Update* (**Figura 15**), o método que escolhemos não recebe mais como parâmetro os valores *Num_Edicao*, *Capa* e *Nivel*, mas um objeto do tipo *Revista* que já contém todos esses valores representados na forma de atributos.

O último passo é avisar o *ObjectDa-*

taSource sobre o tipo de dado que estamos trabalhando. Isso é feito através da sua propriedade *DataObjectTypeName*, que deve ter seu valor configurado para *DTO.Revista*.

> **Nota**: É necessário dar um *Refresh Schema* no *GridView* e *DetailsView*, ou mesmo redefinir a propriedade *DataSourceID*, para que os campos reflitam os atributos da classe de negócio e não mais os campos da tabela.

Pronto! Você já pode executar a aplicação e desfrutar de todas as vantagens de uma solução em camadas 100% OO.

Incluindo regras de negócio

Vamos agora personalizar nossa *BLL* para que inclua regras de negócio. Vamos supor a seguinte regra. O índice que indica o nível de uma revista deve estar entre 1 e 3, porém, a revista tem que possuir um nível balanceado.

Digamos, a título de exemplo, que o nível deve estar entre 1.5 e 2.5, caso contrário ela estará com nível muito avançado ou muito introdutório. A **Listagem 7** mostra como incluir essa validação na *BLL*, para o *Insert* (faça o mesmo para o *Update*).

Note que usamos um *raise* para levantar uma exceção, essa é a forma correta de levantar exceções em objetos de negócio. Jamais use caixas de mensagem ou *JavaScript*, pois isso depende do tipo de interface. A exceção levantada será propagada na pilha até o método que chamou o código, que por sua vez deve dar o tratamento adequado.

Quando usamos o *DataSources*, como o *ODS*, podemos manipular os eventos *Inserted* ou *Updated* para capturar possíveis exceções que tenham ocorrido durante a atualização. É o que fiz na **Listagem 8**. Note que o *ASP.NET AJAX* se encarrega de receber a *exception* e gerar uma caixa de erro ao invés da tradicional página de exceção do *ASP.NET* (**Figura 16**).

Poderíamos implementar também essa mesma regra de negócio no próprio *BD*, se essa for a sua necessidade. Por exemplo, a **Listagem 9** mostra como fazer essa mesma validação na *procedure* de inserção. Testando a aplicação, note que a mensagem de erro é exibida sempre da mesma forma, não importa onde a exceção seja levantada.

E finalmente, poderíamos implementar a regra de negócio no próprio modelo, usando o *set* do atributo *Nivel*. Veja como ficaria a mesma validação agora implementada na entidade (**Listagem 10**).

É claro, essa validação poderia ainda ser feita na camada cliente, usando controles de validação. A questão aqui é que você tem inúmeras opções para escolher onde colocar determinada regra de negócio, analisando suas vantagens e desvantagens.

Conclusão

A solução que apresentei aqui se aplica à maioria dos projetos com *ASP.NET*, permitindo produtividade, modularização, flexibilidade, independência, ao mesmo tempo que desfruta dos melhores recursos da *POO* e do *ASP.NET 2.0*.

Dependendo do tipo de aplicativo que for desenvolver, uma determinada arquitetura deve ser escolhida, depende de você desenvolvedor analisar o que é melhor para o seu projeto.

Desejo sucesso a todos nos projetos *ASP.NET* com *POO* e Multicamadas. Um abraço e até a próxima edição! ●

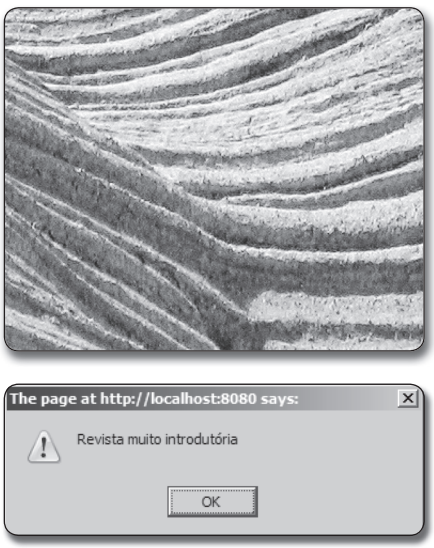


Figura 16. Mensagem de erro

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Listagem 8. Tratando uma exceção na interface cliente

```
procedure TWebForm1.ObjectDataSource1_Inserted( sender: System.Object;
  e: System.Web.UI.WebControls.ObjectDataSourceStatusEventArgs);
begin
   if e.Exception <> nil then
   begin
      e.ExceptionHandled := true;
      raise Exception.Create(e.Exception.InnerException.Message);
   end;
end;
```

Listagem 9. Regra de negócio implementada em Stored Procedure

```
ALTER procedure [dbo].[InsertRevista]
  @NUM_EDICAO int,
  @CAPA varchar(250),
  @NIVEL decimal(18,2)
as
begin
  if @NIVEL < 1.5
    RaisError ('Revista muito avançada', 16, 1);
  else
  if @NIVEL > 2.5
    RaisError ('Revista muito introdutória', 16, 1);
  else
    insert into REVISTAS values (@NUM_EDICAO, @CAPA, @NIVEL)
end
```

Listagem 10. Regra de negócio implementada no modelo

```
property NIVEL: double read FNIVEL write SetFNIVEL;
...
procedure Revista.SetFNIVEL(const Value: double);
begin
  FNIVEL := Value;
  if FNIVEL > 2.5 then
     raise Exception.Create('Revista muito avançada');
  if FNIVEL < 1.5 then
     raise Exception.Create('Revista muito introdutória');
end;
```

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Porta Paralela com Delphi

Entendendo e controlando os registros da porta paralela

Resumo DevMan

Aqui você aprenderá um pouco sobre os conceitos que envolvem a porta paralela do computador e também como construir aplicações que fazem uso dela.

Nesse artigo veremos

- Arquitetura da Porta Paralela;
- Modos de Operação;
- Como programar a porta paralela com o Delphi;
- Quais são os registros da porta e como controlá-los utilizando o componente LPTPort;
- Como listar as portas LPT do PC;
- Como listar os endereços das portas LPT do PC;
- Como ler e escrever valores nos registros das portas paralelas do PC.

Qual a finalidade

- Controle e desenvolvimento de aplicações que utilizem a porta paralela.

Quais situações utilizam esses recursos?

- Projetos eletrônicos, controle de dispositivos e comunicações de dados.

A porta paralela é uma velha conhecida de todos nós, que era muito usada para conectar uma impressora ao PC, e hoje se você reparar na parte traseira do seu micro, para aqueles que não possuem micros tão novos, há uma porta com 25 pinos com um pequeno símbolo de impressora. Esta porta é conhecida como porta LPT ou porta de impressora. Atualmente esta porta é utilizada para controle de dispositivos e/ou transferência de dados, em projetos de robótica, eletrônica e até controle de displays LCD. O primeiro passo, para explorar a porta paralela é aprender como se beneficiar da porta em suas aplicações diárias, apesar de que, na maioria dos casos, ela será utilizada em projetos eletrônicos devido a sua facilidade de controle.

Alguns dos pré-requisitos básicos e importantes com relação à porta paralela para que possamos tirar máximo proveito dela, são:

- Saber como acessá-la;
- Quando utilizar os seus modos de operação;

Thiago Batista Limeira
tblimeira@gmail.com
é engenheiro da computação e trabalha com Delphi há mais de 4 anos, atualmente desenvolve programas e interfaces homem-máquina que envolvam comunicação de dados. MSN: tblimeira@hotmail.com

- Como funcionam seus registros;
- Etc.

Lembro-me de que anos atrás, durante um de meus projetos da faculdade, que exigia a utilização da porta paralela, tive muita dificuldade para encontrar material sobre o assunto do ponto de vista de um programador *Delphi*, ou seja, como acessar e controlar a porta através dele. Este artigo tem por objetivo desvendar, ainda que de forma bem rápida e prática, como funciona a porta paralela juntamente com seus registros e modos de operação. Mostrarei também os conceitos básicos de sua arquitetura e alguns princípios de programação utilizando *Delphi*. E para finalizar, faremos uma aplicação onde poderemos ler e escrever valores aos registros da porta paralela.

Conceitos da Porta Paralela

A porta paralela inicialmente foi desenvolvida para conectar uma impressora ao PC, porém podemos programar esta porta para muitas outras aplicações.

Ela pode ser utilizada por duas razões principais: *controle de dispositivos* e *comunicação*. Podemos programar esta porta para ambas as funções. As portas paralelas são mais fáceis de programar e mais rápidas em comparação às portas seriais. Mas a principal desvantagem é que necessita de mais fios para transmitir os dados (9 em comparação aos 3 fios da porta serial). Por este motivo, as portas paralelas não são utilizadas em comunicações a muita distância. A seguir veremos as diferenças básicas de funcionamento entre a porta paralela e a serial.

Na serial temos dois fios de dados: um para *transmissão* e outro para *recepção*. Para enviar os dados através da porta serial, enviamos um *bit* após o outro ("daí a origem do nome serial, os *bits* são transmitidos em série"), com alguns *bits* extras como o de início (*start bit*), *bit* de parada (*stop bit*) e o *bit* de paridade (*parity bit*) para detecção de erros.

Já na porta paralela, todos os oito *bits* do *byte* serão enviados simultaneamente e uma indicação enviada em outro fio. Teremos ainda alguns fios de transmissão de dados, alguns de controle, e outros de *handshaking*, que é o processo de reconhecimento entre dois dispositivos, máquinas, micros etc. (**Figura 1**).

Normalmente na parte traseira do micro existe um conector DB25 fêmea com 25 pinos (**Figura 2**). A **Tabela 1** apresenta os pinos e suas respectivas funções.

Modos de operação

A porta paralela no modelo original do IBM PC, e/ou qualquer outra porta que simule o modelo original da porta, é normalmente chamado de SPP ou Porta Paralela Padrão (*Standard Parallel Port*). Esta porta inicialmente era baseada em uma interface de impressora, neste modo de operação a porta paralela pode transmitir oito *bits* (*1 byte*) simultâneos para algum periférico e também é possível trabalhar transmitindo quatro *bits* (*1 nibble*), apesar de mais lento. Este modo possui taxa de transferência de 150 KB/s e é unidirecional.

O modo EPP ou Porta Paralela Avançada (*Enhanced Parallel Port*) foi originalmente desenvolvido pela *Intel*. Neste modo é possível ler ou escrever um *byte*, e também alternar direções rapidamente, por isso é muito eficiente para utilização com dispositivos que transferem dados em ambas as direções. O EPP pode também simular um SPP e este modo de porta paralela é bidirecional e atinge uma taxa de transferência de 2 MB/s. Entretanto, para atingir esta taxa, necessita de um cabo especial, pois o cabo tradicional só comporta uma taxa de até 150 KB/s.

E o último modo, o ECP (*Extended Capabilities Port*) foi proposto pela *Hewlett Packard* e *Microsoft*. Ela é igual ao EPP, porém utiliza um canal de DMA (*Direct Memory Access*), que faz com que a transmissão e recepção sejam feitas sem a intervenção do processador, aumentando o desempenho do micro. Transmissões ECP são muito úteis para impressoras,

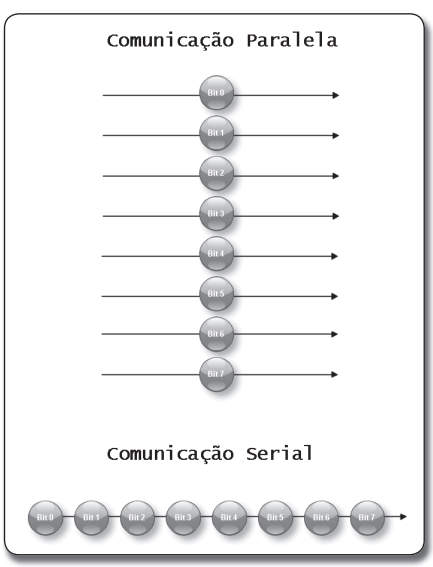


Figura 1. Diferenças de transmissão de dados entre a porta paralela e a serial

Figura 2. Uma porta paralela na parte traseira de um notebook (conector padrão DB25 fêmea)

Pino	Descrição	Direção
1	Strobe	Entrada/Saída
2 a 9	Data	Entrada/Saída
10	Busy	Entrada
11	Out of Paper	Entrada
12	Select	Entrada
13	Auto feed	Entrada
14	Error	Entrada/Saída
15	Dados	Entrada
16	Select In	Entrada/Saída
17	Init	Entrada/Saída
18 a 25	Ground	Terra

Tabela 1. Pinagem e Funções Conector DB25

scanners, e outros periféricos que transmitem grandes blocos de dados. O ECP também pode simular o modo SPP.

Organização da Porta Paralela

Agora vamos compreender como as portas são organizadas. Existem três registros associados com a porta LPT:

- O registro de dados;
- Controle;
- Status.

O registro de dados cuida dos pinos de dados da porta, isto significa que se armazenarmos um byte de dados no registro de dados, estes por sua vez podem ser enviados aos pinos de dados da porta;

Normalmente, os registros de dados, controle e status terão os endereços mostrados na **Tabela 2**, vamos precisar destes endereços mais tarde durante a programação.

Para obter mais detalhes das portas paralelas do seu computador entre no Painel de *Controle>Sistema>Hardware>Gerenciador* de *Dispositivos>Portas(COM&LPT)>Porta da Impressão ECP(LPT1)* e clique duas vezes (**Figura 3**). Selecione a aba recursos, onde você encontrará o endereço da porta paralela (**Figura 4**). Observe que o primeiro valor listado na linha *Intervalo de E/S* é o endereço base da porta paralela do seu computador.

Programando a Porta Paralela

Por questões de segurança as versões do *Windows NT/2000* e *XP* não permitem o acesso direto ao *hardware*, isto é, não permitem a direta manipulação dos recursos de *hardware* tais como a porta paralela, por isso é necessário um *driver* que "converse" com o núcleo (*Kernel*) desse sistema para ter acesso às portas físicas do computador. Nas versões do *Windows 95/Me* e *98* não há restrição e o acesso pode ser direto, portanto, não há necessidade do uso de *drivers*.

Sabemos que a utilização de *dlls* gera certo desconforto, pois as mesmas devem ser distribuídas juntamente com a aplicação para que a mesma funcione, porém utilizaremos um componente gratuito de acesso à porta paralela que armazena *drivers* de acesso para as versões do *Windows 95/Me/98/NT/2000* e *XP* e que se encarrega de utilizar *drivers* de acesso ao *hardware*, ou seja, ele se encarrega de carregar e descarregar da memória as *dlls* necessárias para manipulação da porta. O referido componente encontra-se disponível para download junto com o código-fonte deste artigo.

Faremos uma aplicação simples para demonstrar a detecção das portas paralelas, manipulação dos pinos de dados e controle da paralela além da leitura dos registros de dados, status e controle.

Construindo o exemplo

Inicialmente abra o *Delphi* e crie uma nova aplicação. Salve o projeto como *ControleLPT.dpr* e a *Unit* do formulário principal com o nome *uMain.pas*. Coloque no formulário criado um *LPTPort*. Agora dê um duplo clique no formulário e no evento *OnCreate* escreva o seguinte código:

```
LPTPort1.SelectPort($378);
```

Com isso selecionamos o endereço base da porta paralela que é padrão nos computadores. Conforme citado anteriormente é o endereço 378 em hexadecimal (888 em decimal), em *Delphi* devemos colocar o símbolo **$** para que o compilador entenda que se trata de um número hexadecimal. Coloque no formulário um *GroupBox* (*grbPinos*) e cinco *labels* sobre o *grbPinos* .Modifique suas propriedades conforme a **Tabela 3.**

Estes *Labels* serão utilizados para ler os valores em decimal e binário do registro de dados da porta, utilizando o método *MostrarEstado* que está no código a seguir:

```
lblDec.Caption := IntToStr(LPTPort1.Inp);
lblBin.Caption := DecToBin(LPTPort1.Inp);
```

A rotina *MostrarEstado* faz a leitura do registro selecionado através da função *Inp* e apresenta o valor no *Caption* dos *Labels*, repare que o componente possui uma ro-

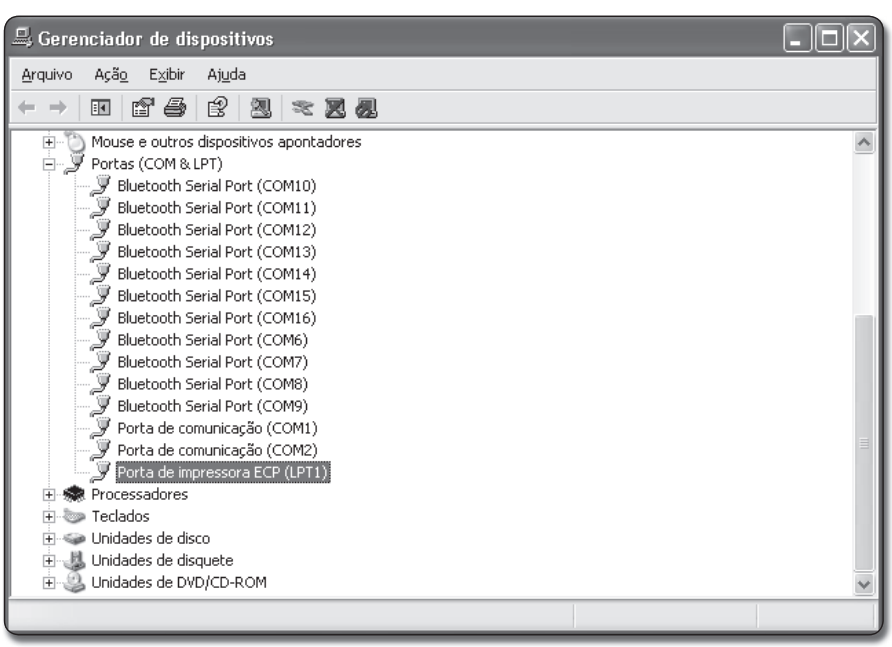


Figura 3. Gerenciador de Dispositivos: Portas (COM&LPT)

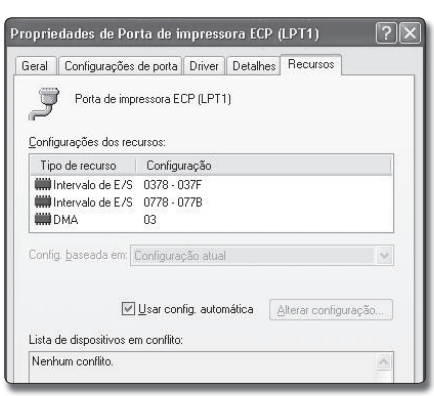


Figura 4. Propriedades da Porta de Impressora no Windows

Registro	Endereço LPT1
Registro de Dados	Endereço Base (378H) + 0
Registro de Status	Endereço Base (378H) + 1
Registro de Controle	Endereço Base (378H) + 2

Tabela 2. Registros e endereços

tina de conversão *DecToBin* que converte um valor decimal em binário. Declare então essa *procedure* na sessão *public* do formulário e em seguida pressione *Ctrl + Shift + C* para que o *Delphi* crie o cabeçalho. A assinatura completa da *procedure* você pode ver em seguida. Nela digite o código que vimos anteriormente:

```
procedure MostrarEstado;
```

Agora insira um *RadioGroup*(*rdgRegistros*) e insira na propriedade *Items* três linhas de texto: "Registro de Dados", "Registro de Status" e "Registro de Controle". Agora dê um duplo clique no *rdgRegistros* e digite o código da **Listagem 1** em seu evento *OnClick*.

Feito isto, coloque oito *CheckBoxes* no *grbPinos* e modifique a propriedade *Name* de cada um para "D0", "D1", "D2", "D3", "D4", "D5", "D6" e "D7", respectivamente. Adicione a sigla *Chk* ao *Name* de cada *CheckBox* e use o texto como *Caption* dos componentes, ou seja, *ChkD0..ChkD7*. Por fim, altere para *False* a propriedade *Enable* de cada *CheckBox*.

Estes *CheckBoxes* irão simular os pinos de dados da porta paralela, como estes são os pinos mais utilizados, o componente possibilita acessá-los diretamente através do seguinte método:

```
SetPin(Name: string; State: Boolean);
```

Onde devemos passar como parâmetros, uma *String* com o nome do pino que desejamos utilizar e no *State* um parâmetro *Booleano* que é o estado que queremos colocar no pino.

Como estamos alternando entre os registros, precisaremos de uma rotina para inibir os *CheckBoxes* quando estes não estiverem sendo utilizados pelos registros que estamos selecionando no programa. Para isso crie uma nova *procedure* com nome e assinatura proposto a seguir e codifique-a como na **Listagem 2**:

```
SetPinOutCheckBoxState(State: Boolean);
```

Para entendermos, apenas estamos fazendo um laço *for* para localizar os *CheckBoxes* em tela e ativamos/desativamos de acordo com o parâmetro que chega.

Agora vamos inserir três *Buttons* (*btnAbrir, btnEnviar* e *btnLer*). Modificaremos a propriedade *Enable* destes botões para *False*. Troque o *Caption* de cada um para "Fechar", "Enviar" e "Ler", respectivamente. Dê um duplo clique no *btnAbrir* e no evento *OnClick* digite o código da **Listagem 3**.

Utilizando a propriedade *Tag* deste botão que inicialmente é igual a *0*, estamos definindo o *port* da porta paralela, que é o 378 hexadecimal. Além disso, através da função *Open* do *LPTPort* o componente carregará as *Dlls* responsáveis pelo acesso ao *hardware* e retornará o valor *True* caso tenha obtido sucesso, o clique do botão também irá ativar os botões *btnEnviar* e *btnLer*. Por isso mantenha a propriedade *Enabled* dos demais botões para *False*. Este botão ainda ativará os *CheckBoxes* correspondentes aos pinos.

Também preparamos para que o *btnAbrir* feche o componente *LPTPort* e descarregue de memória as *dlls* caso um segundo clique seja dado no botão.

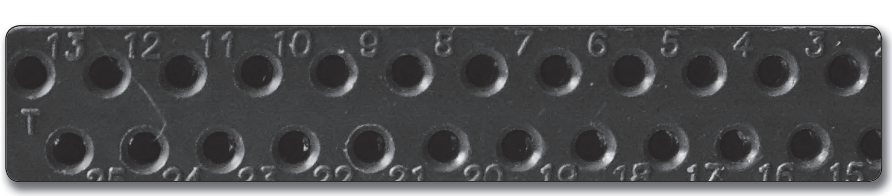
Voltando agora aos *CheckBoxes*, iremos construir uma função que verificará o estado da propriedade *Checked* e irá refletir

Label	Caption	Name
Label1	LPT Port	lblPortHeader
Label2	Dec	lblPort
Label3	Decimal	lblDecHeader
Label4	----------	lblDec
Label5	Binário	lblBinHeader
Label6	----------	lblBin

Tabela 3. Tabela Labels

Listagem 1. Código de seleção de registros da Porta Paralela

```
procedure TForm1.rdgRegistrosClick(Sender: TObject);
begin
  case rdgRegistros.ItemIndex of
    0:
      begin
        SetPinOutCheckBoxState(True);
        btnEnviar.Enabled := True;
        btnLer.Enabled := True;
        lblBinHeader.Enabled := True;
        lblBin.Enabled := True;
        lblDecHeader.Enabled := True;
        lblDec.Enabled := True;
        edtEnviar.Enabled := True;
        btnEnviar.Enabled := True;
      end;
    1:
      begin
        SetPinOutCheckBoxState(False);
        btnEnviar.Enabled := False;
        btnLer.Enabled := True;
        lblBinHeader.Enabled := False;
        lblBin.Enabled := False;
        lblDecHeader.Enabled := False;
        lblDec.Enabled := False;
        edtEnviar.Enabled := False;
        btnEnviar.Enabled := False;
      end;
    2:
      begin
        SetPinOutCheckBoxState(False);
        btnEnviar.Enabled := True;
        btnLer.Enabled := True;
        lblBinHeader.Enabled := False;
        lblBin.Enabled := False;
        lblDecHeader.Enabled := False;
        lblDec.Enabled := False;
        edtEnviar.Enabled := True;
        btnEnviar.Enabled := True;
      end;
  end;
end;
```



este estado aos pinos de dados da porta. Para isso declare a *procedure OnPinOutClick* na sessão *public* e digite o código da **Listagem 4.** A procedure utiliza o método *SetPin* do componente *LPTPort* para configurar o *Pino* desejado da porta paralela. Observe que estamos passando *True* ou *False* conforme o estado do *CheckBox*. Veja o trecho de código:

```
case C.State of
  cbChecked: LPTPort1.SetPin(Pin, True);
  cbUnChecked: LPTPort1.SetPin(Pin, False);
end;
```

É possível listar todas as portas *LPT* do computador utilizando o *LPTPort*, para isso vamos inserir um *ComboBox* sobre o *grbPinos* (*cbLPTPort*) e no evento *OnDropDown* colocaremos o seguinte código:

```
LPTPort1.EnumPorts(cbLPTPort.Items);
```

O método *EnumPorts* recebe um parâmetro do tipo *TStrings*. Este método lista todas as portas *LPT* disponíveis no computador e armazena esta lista no parâmetro informado. No nosso caso a propriedade *Items* do *cbLPTPort*.

É possível também determinar o endereço da porta paralela, através da propriedade *Ports* do componente. Esta propriedade é uma lista de todas as portas paralelas do computador, e através desta lista podemos recuperar o nome e o endereço da porta, utilizando respectivamente as propriedades *Name* e *Port* da lista *Ports*, isto está sendo feito no evento *OnClick* do *cbLPTPort* conforme o código a seguir:

```
  if cbLPTPort.ItemIndex = -1 then
    Exit;
  LPTPort1.SelectPortByIndex(cbLPTPort.
ItemIndex);
  lblPort.Caption := Format('%s = 0x%.3xH',
    [LPTPort1.Ports[cbLPTPort.ItemIndex].
Name,
    LPTPort1.Ports[cbLPTPort.ItemIndex].
Port]);
```

Listagem 2. Habilitar e desabilitar os CheckBoxes

```
procedure TForm1.SetPinOutCheckBoxState(State:
  Boolean);
var
  I: Integer;
  C: TCheckBox;
begin
  for i := 0 to ComponentCount - 1 do
  begin
    if Components[i] is TCheckBox then
    begin
      C := TCheckBox(Components[i]);
      if Copy(UpperCase(C.Name), 1, 4) = 'CHKD' then
        C.Enabled := State;
    end;
  end;
end;
```

Listagem 3. Código do evento btnAbrirClick

```
procedure TForm1.btnAbrirClick(Sender: TObject);
begin
  case btnAbrir.Tag of
    0:
      begin
        btnAbrir.Caption := 'Fechar';
        btnAbrir.Tag := 1;
        LPTPort1.LPTPort := $378;
        if LPTPort1.Open then
        begin
          SetPinOutCheckBoxState(True);
          cbLPTPort.Enabled := True;
          btnEnviar.Enabled := True;
          btnLer.Enabled := True;
        end;
      end;
    1:
      begin
        btnAbrir.Caption := 'Abrir';
        btnAbrir.Tag := 0;
        LPTPort1.Close;
        SetPinOutCheckBoxState(False);
        cbLPTPort.Enabled := False;
        btnEnviar.Enabled := False;
        btnLer.Enabled := False;
      end;
  end;
end;
```

Listagem 4. Código para ativar os pinos de dados da porta através dos Checkboxes

```
procedure TForm1.OnPinOutClick(Sender: TObject);
var
  C: TCheckBox;
  Pin: string;
begin
  if Sender is TCheckBox then
  begin
    C := (Sender as TCheckBox);
    Pin := Format('D%u', [C.Tag]);
    case C.State of
      cbChecked: LPTPort1.SetPin(Pin, True);
      cbUnChecked: LPTPort1.SetPin(Pin, False);
    end;
    MostrarEstado;
  end;
end;
```

Escrevendo valores pela porta LPT

Agora vamos construir a rotina de escrita na porta paralela, que o *LPTPort* facilita e muito, mas antes para finalizar necessitamos colocar no formulário dois *Edits* (*edtEnviar* e *edtLer*). Estes *Edits* serão utilizados para escrever e ler valores na porta LPT. Feitas todas as alterações do formulário, teremos uma tela parecida com a **Figura 5**.

Dê um duplo clique no botão *btnEnviar* e em seu evento *OnClick* digite o código da **Listagem 5**, este código contém a rotina de envio de dados para o *registro selecionado*. Primeiramente recuperamos o valor que está contido no *edtEnviar* utilizando a função *Trim*. E utilizando a função declarada na *Unit* do próprio *LPTPort*, com a função *IsNumeric*, verificamos se a *String* contida no *edtEnviar* realmente é um valor numérico, caso contrário, mostramos uma mensagem de aviso.

Com isso, de acordo com o *RadioButton* selecionado, escolhemos o registro com o qual desejamos trabalhar, (lembrando mais uma vez que o registro de *Status* no endereço 379H é apenas para leitura). Feita esta verificação convertemos a *String* em um valor inteiro e enviamos para a porta paralela utilizando o método *Out* do componente. Caso o registro selecionado seja o de dados, nós atualizamos o valor binário e decimal nos *labels lblDec* e *lblBin*, utilizando o método *MostrarEstado*.

Note que podemos enviar para o registro de dados um valor máximo de 255,

devido a cada pino representar um *bit* tendo um total de oito *bits* (*1 byte*) que pode variar de 0 a 255.

Lendo valores da porta LPT

Agora que já sabemos como enviar valores para a porta paralela, estaremos abordando como ler valores. Dê um duplo clique no *btnLer* e digite o código da **Listagem 6** no evento *OnClick* do botão.

Repare no código que selecionamos o endereço base, no caso, o 378H e dependendo do registro selecionado no comando *case*, estamos alterando o valor da propriedade *InpOffset* do *LPTPort*. Esta propriedade armazena um valor numérico, que seleciona qual registro estará sendo lido pelo método *Inp* do componente. Por isso, se desejamos ler o registro de controle, devemos sempre selecionar o endereço base (378H no nosso caso) e modificar a propriedade *InpOffset* para o valor 2. No nosso exemplo estamos lendo o registro selecionado e mostrando este valor no *edtLer*.

Compile a aplicação e efetue todos os testes. Veja que tudo funciona perfeitamente e sem quase nenhum esforço.

Conclusão

Pudemos ver neste artigo que, através do componente *LPTPort*, fica muito fácil desenvolver programas que acessem à porta paralela sem a necessidade de distribuição de *dlls*. Deste modo, poderemos agora aproveitar aqueles computadores que possuem alguma porta paralela para desenvolver o controle de dispositivos ou projetos eletrônicos. Fiquem com Deus, um forte abraço, e até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

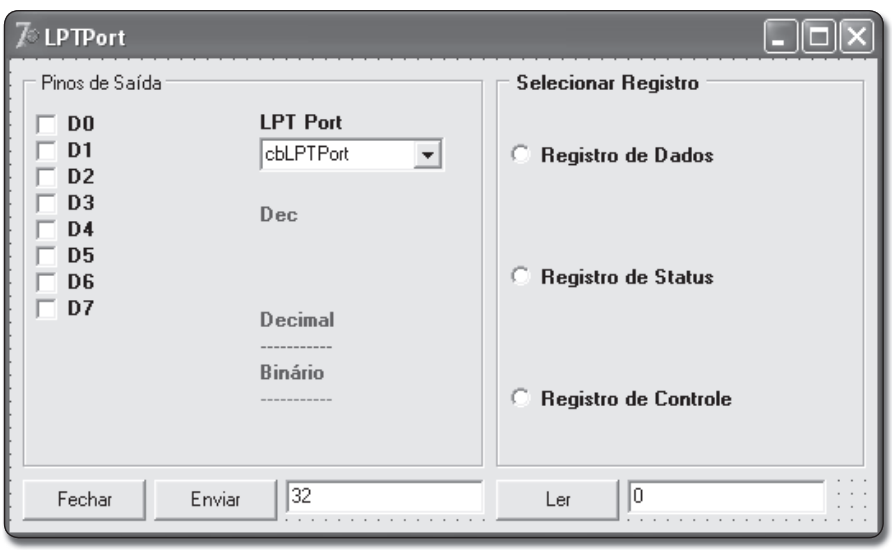
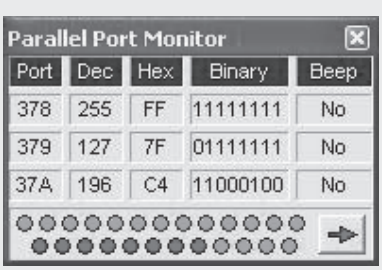


Figura 5. Sugestão de GUI (Graphic User Interface) para a aplicação

Nota do DevMan

O desenvolvimento de programas que acessem à porta paralela pode ser uma tortura para aqueles que não têm conhecimentos básicos de eletrônica, pois a medição de tensão dos pinos facilita a visualização do que está de fato ocorrendo com os pinos da porta. Existe uma ferramenta de monitoramento da porta paralela que possibilita a visualização gráfica do estado dos pinos e a leitura e/ou manipulação dos valores dos registros da porta, chamada *Parallel Port Monitor*. Seu uso é realmente recomendado durante o desenvolvimento de quaisquer projetos de programas que utilizem a porta paralela. Esta ferramenta pode ser baixada gratuitamente em: *www.geekhideout.com/downloads/parmon.zip*



Listagem 5 Código efetuar escrita de valores na porta LPT (btnEnviarClick)

```
procedure TForm1.btnEnviarClick(Sender: TObject);
var
  S: string;
begin
  S := Trim(edtEnviar.Text);
  if not (IsNumeric(S)) then
  begin
    MessageDlg(Format('Atenção valor %s incorreto!', [S]),
      mtWarning, [mbOk], 0);
    exit;
  end;
  case rdgRegistros.ItemIndex of
    0: LPTPort1.SelectPort($378);
    2: LPTPort1.SelectPort($37A);
  end;
  LPTPort1.out(StrToInt(S));
  if rdgRegistros.ItemIndex = 0 then
    MostrarEstado;
end;
```

Listagem 6. Código efetuar leitura de valores na porta LPT (btnLerClick)

```
procedure TForm1.btnLerClick(Sender: TObject);
begin
  LPTPort1.SelectPort($378);
  case rdgRegistros.ItemIndex of
    0: LPTPort1.InpOffset := 0;
    1: LPTPort1.InpOffset := 1;
    2: LPTPort1.InpOffset := 2;
  end;
  edtLer.Text := IntToStr(LPTPort1.Inp);
end;
```

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Ferramenta para vários servidores de banco de dados

Ferramentas para exportação, importação e administração de banco de dados

Luciano Pimenta
lucianoalmeidapimenta@gmail.com
é Técnico em Processamento de Dados, desenvolvedor Delphi/C#. Palestrante da 4ª edição da Borland Conference (BorCon). Autor de mais de 50 artigos e de mais de 250 vídeo aulas publicadas em revistas e sites especializados. Atualmente é programador Web da FullSoft - Mobile Solutions em Caxias do Sul-RS. Blog: lucianopimenta.blogspot.com. Site: www.lucianopimenta.net

Resumo DevMan

A intenção desse artigo é mostrar uma ferramenta super completa para utilização no dia-a-dia. Essa ferramenta pode ser utilizada para dar manutenção em vários bancos de dados. Veremos o SQL Studio.

Nesse artigo veremos

- Utilização de ferramentas para manipulação de bancos de dados;
- Principais funcionalidades e facilidades do SQL Studio.

Qual a finalidade

- Facilitar testes e manipulação de dados no dia-a-dia.

Quais situações utilizam esses recursos?

- Podemos utilizar programas para manipulação de dados em todas as situações possíveis, incluindo testes de performance e do desenvolvedor.

Não é novidade empresas possuírem vários servidores de banco de dados para suas aplicações, por serem projetos legados ou por simples opção. Imagine uma empresa com um sistema com *Firebird* e outros com *SQL Server*, por exemplo.

Quando necessário você precisa dar manutenção no banco, e para isso utiliza, por exemplo, o *IBExpert* para *Firebird* e outra ferramenta para o *SQL Server*. A maioria das ferramentas são parecidas, mas cada uma tem sua características de funcionalidades, para exportação, importação, inserção de dados etc.

Mas, e se você utilizar uma ferramenta para vários tipos de servidores, mas com as mesmas características? No caso, não estamos falando de apenas uma ferramenta, mas sim uma para cada servidor, mas com as mesmas funcionalidades.

É isso que temos com as ferramentas da *SQL Manager* (*www.sqlmanager.net*), onde a mesma suporta: *MySQL*, *SQL Server*, *PostegreSQL*, *InterBase/Firebird*, *Oracle*, *DB2* e *DBISAM* (**Figura 1**), onde as características das ferramentas são as mesmas, para criação de objetos, exportação, importação etc.

Figura 1. Suporte a vários tipos de servidores de banco de dados

Cada *Studio* como é apresentado, possui ferramentas auxiliares que podem ser baixadas ou adquiridas separadamente, como por exemplo, uma ferramenta apenas para importar/exportar dados, para extrair estrutura e dados do banco, migrar de um banco para outro (*pump*) entre outros (**Figura 2**).

Temos ainda componentes para aplicar as funcionalidades das ferramentas dentro de projetos utilizando *Delphi* e *C++*, uma boa dica, mas que não será tratada nesse artigo. Na **Figura 3** temos um exemplo da funcionalidade (gerador de planilhas do *Excel*) que podemos colocar em projetos.

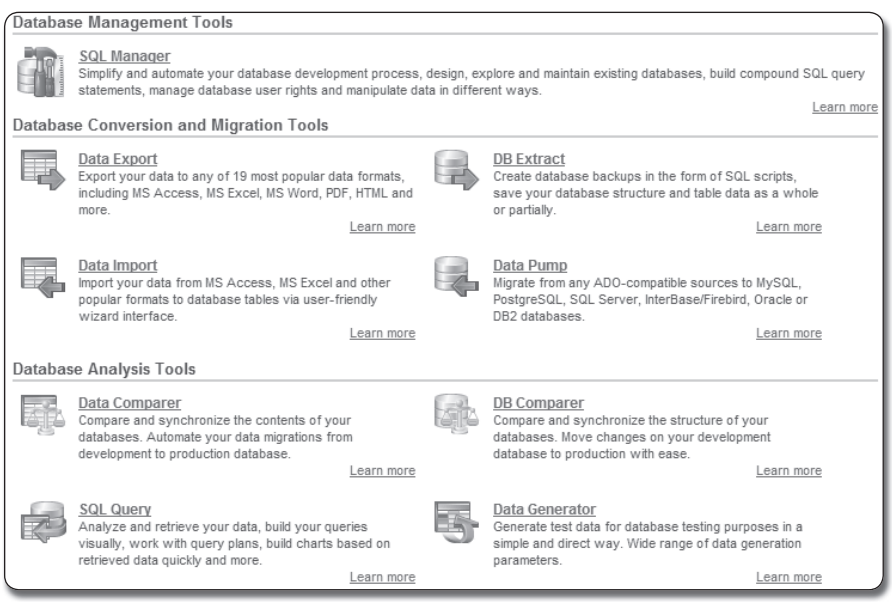


Figura 2. Ferramentas do Studio

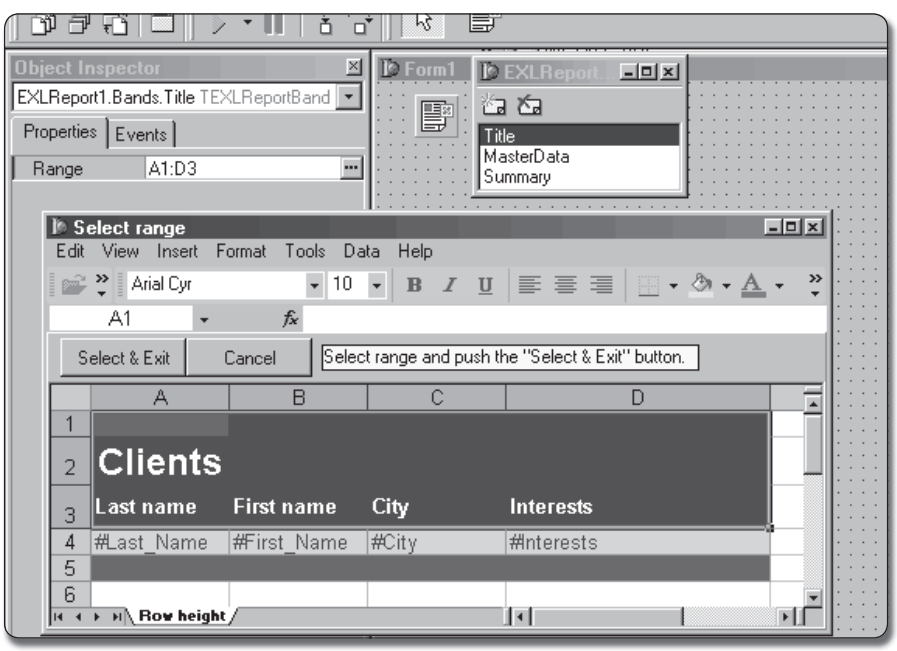


Figura 3. Componentes VCL para criar funcionalidades da ferramenta

Instalação

Vamos mostrar a ferramenta para *SQL Server* (o *SQL Studio for SQL Server*), mas fique à vontade para utilizar outro. Você pode baixar uma versão *trial* no site mencionado anteriormente.

A instalação é simples e não necessita de configuração muito avançada. Após a instalação, abra a ferramenta, através do atalho criado na área de trabalho. Ao abrir o *SQL Studio* é mostrado, ao lado esquerdo, um painel com as opções de ferramentas, tarefas, acesso a suporte da empresa e os bancos de dados registrados (**Figura 4**).

Na opção *Tools*, temos quatro opções com as seguintes características:

- *DataBase Management*: concentra as ferramentas para manipulação dos objetos do banco. Temos o *SQL Manager* onde

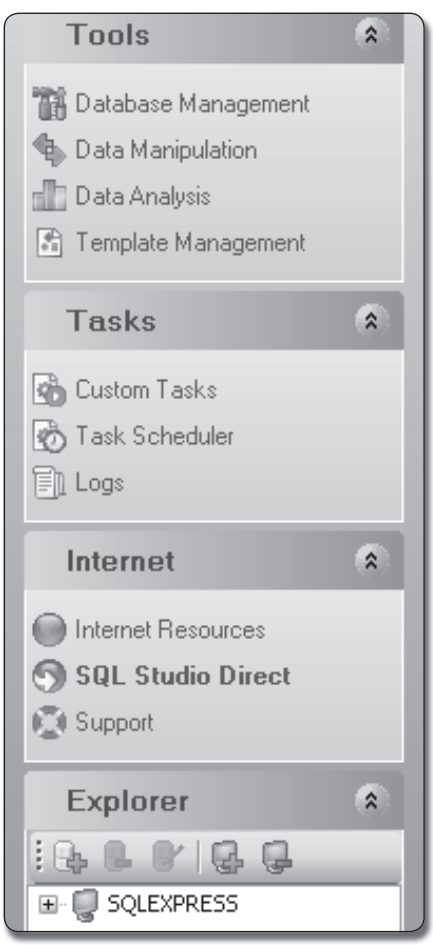


Figura 4. Painel com opções

podemos criar bancos, tabelas entre outros objetos, onde se concentra todas as funcionalidades de administração do banco de dados. Temos ainda nessa opção, as funcionalidades de *backup/restore*, atachar/desatachar bancos de dados;

- *Data Manipulation*: concentra as ferramentas para manipular dados para exportação/importação, gerador de *script* (*SQL Script*) e de dados para testes (*Data Generator*) e também uma ferramenta para migração de bancos para *SQL Server* (*Data Pump*);

- *Data Analysis*: concentra as ferramentas para analisar o banco de dados, como: comparador de dados (*Data Comparer*) e do banco (*Database Comparer*), extrator de *scripts* do banco (*DB Extract*) e um gerador de consultas para analisar planos, gerar gráficos sobre a performance de sua consutla *SQL*;

- *Template Management*: concentra uma ferramenta para geração de *templates* para salvar suas consultas mais usadas. Os *templates* podem ser salvos das ferramentas de importação/exportação, *backup/restore*, comparador de dados e bancos, gerador de *scripts* etc.

Com o *SQL Manager* temos concentrado a maioria das ferramentas auxiliares, ou seja, através do *SQL Manager*, podemos importar/exportar dados, *backup/restore*, etc. Abra a ferramenta e vamos registrar um banco de dados.

Conhecendo o SQL Manager

Acesse o menu *Database>Register Database* e na primeira tela do *wizard*, escolha o nome do *Server name* (servidor), digite o usuário e senha na autenticação através do *SQL Server*. Se desejar, escolha a opção de autenticação pelo sistema operacional (*Windows*) e clique em *Next*.

Nota: Se você estiver usando o *SQL Server Express*, após o nome do servidor em *Host*, devemos digitar a instância do *SQL Server Express*, que por padrão é "*SQLEXPRESS*". Por exemplo (para conexão local): "*(local)\SQLEXPRESS*".

Na próxima tela, escolha o banco de dados em *Database name* (vamos usar o *Northwind*) e clique em *Finish*. No painel *Databases* temos o banco de dados registrado e para acessar seus objetos, basta dar um duplo clique na opção (**Figura 5**).

Vamos abrir a tabela *Categorys* e veremos através da **Figura 6** que temos uma janela com várias abas.

A primeira aba a ser mostrada quando abrimos a tabela é a dos campos. Nela, podemos adicionar/alterar/excluir campos da tabela. Para acessar as opções, clique com o botão direito no campo e veja o menu. Ao adicionar ou alterar um campo, o editor mostra as opções da **Figura 7**.

No editor definimos o campo como chave primária, se o mesmo será nulo, campo auto-incremento e o tipo do mesmo. Temos a aba *Description*, onde podemos dar uma descrição para o campo, indicando qual seu objetivo na tabela.

Essa descrição é muito interessante, pois podemos fazer uma documentação do banco, colocando em cada campo qual

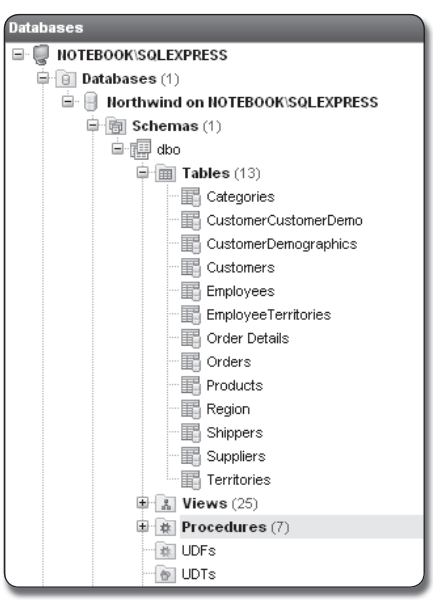


Figura 5. Objetos do banco de dados

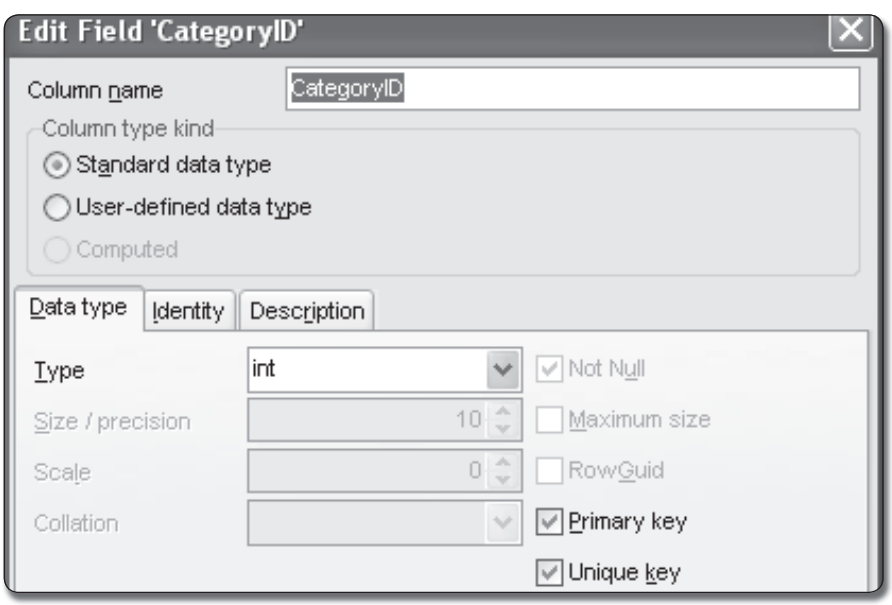


Figura 7. Editor para adicionar ou alterar informações de um campo

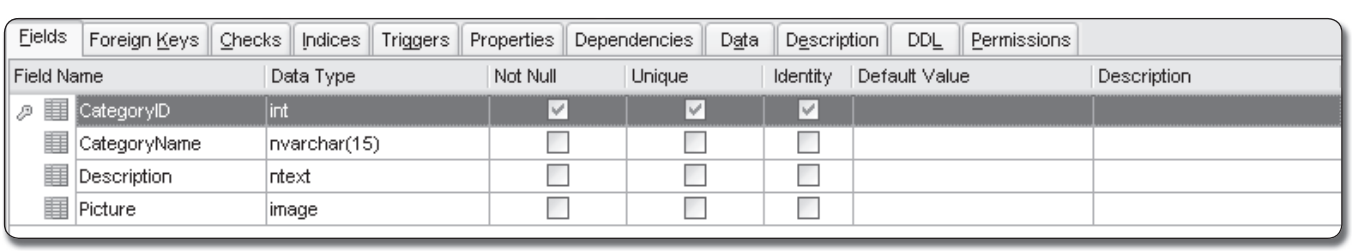


Figura 6. Objeto Table e suas opções

sua finalidade, sendo extremamente útil quando precisarmos fazer manutenção no BD. Essa descrição é válida para todos os objetos do banco (tabelas, *Triggers*, *Stored Procedures* etc).

Veremos adiante neste artigo, como gerar essa documentação em páginas *HTML* onde é possível navegar entre as mesmas.

Relacionamento entre tabelas

A aba *Foreign Keys* permite criar o relacionamento entre as tabelas, as chaves estrangeiras. Clique com o botão direito, escolha *New Foreign Key* e veja o editor visual para criação de chaves estrangeiras (**Figura 8**).

Basta agora, escolher o campo a ser relacionado, a tabela e as opções para atualização e exclusão (*Update action* e *Delete action*). Após, clique em *Compile* e a chave estrangeira estará criada.

As abas *Checks* e *Indices* são semelhantes e de fácil manipulação. Na aba *Triggers* podemos criar os "gatilhos" relacionados com a tabela em questão.

Em *Properties* temos informações da tabela, como: números de registros, espaço usado pelos dados, data de criação, entre outros.

A aba *Dependencies* é bastante útil, pois nela, podemos visualizar quais os objetos dependem da tabela e quais os objetos que a tabela depende. Na **Figura 9** temos as dependências da tabela *Employees*.

Note que em todos os editores, temos uma aba *Description* para adicionarmos a descrição do objeto, a mesma função da aba da tabela *Catogories*. Na aba *DDL* temos todos os comandos vinculados à tabela (criação da tabela, índices, chaves estrangeiras etc).

Na aba *Permissions* podemos atribuir/remover permissões sobre a tabela (basta dar um duplo clique), apenas escolhendo as opções, inclusive em nível de campo (**Figura 10**).

Visualizando os dados

A aba *Data*, com certeza é a mais importante, pois além de mostrar os dados da tabela, tem as opções de manipulação de dados, filtros etc. Uma funcionalidade útil é o agrupamento de registros, onde basta arrastar o campo para a parte superior (em cinza escuro) e pronto, nossos dados estão agrupados.

Para filtrar os dados da tabela, temos várias opções, uma delas, é através de uma lista de seleção que existe em cada campo, onde basta escolher um valor (salvo no respectivo campo, onde são mostrados os valores sem repetição) para a tabela ser filtrada por esse valor.

Ao aplicar esse filtro, na parte inferior, temos uma barra onde é mostrado o(s) filtro(s) utilizado(s) e um botão para acessarmos o editor onde podemos adicionar mais filtros. No editor, temos várias opções de filtros e várias combinações possíveis (**Figura 11**).

O editor é inteligente o suficiente para mostrar os operadores de acordo com o campo, ou seja, se tivermos um campo inteiro, os operadores são os que podem ser usados em campos de tipo inteiro.

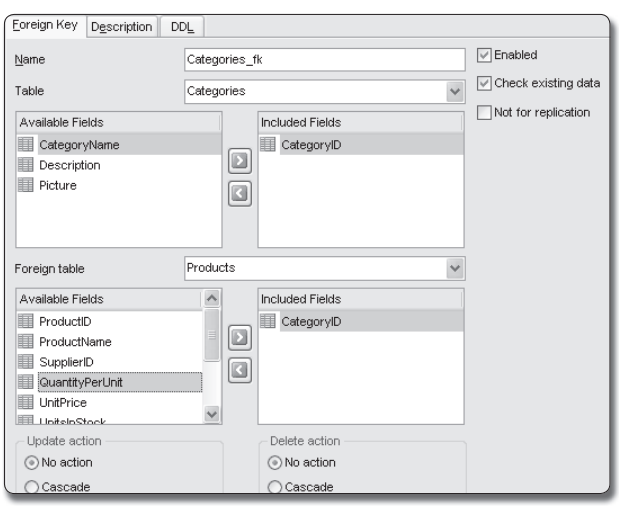


Figura 8. Criação visual de chaves estrangeiras

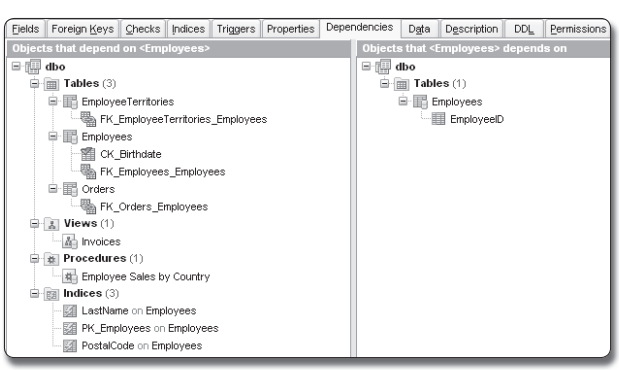


Figura 9. Dependências da tabela Employees

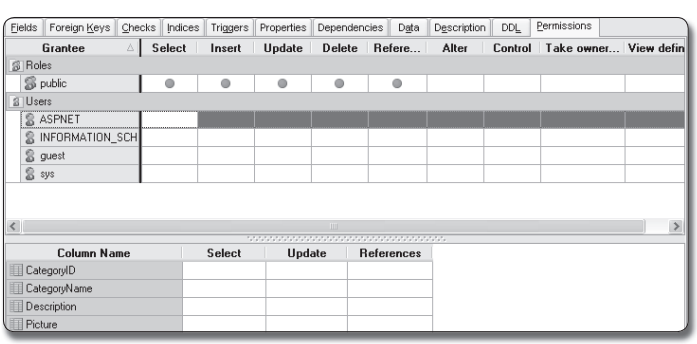


Figura 10. Opções de permissões na tabela e nos campos

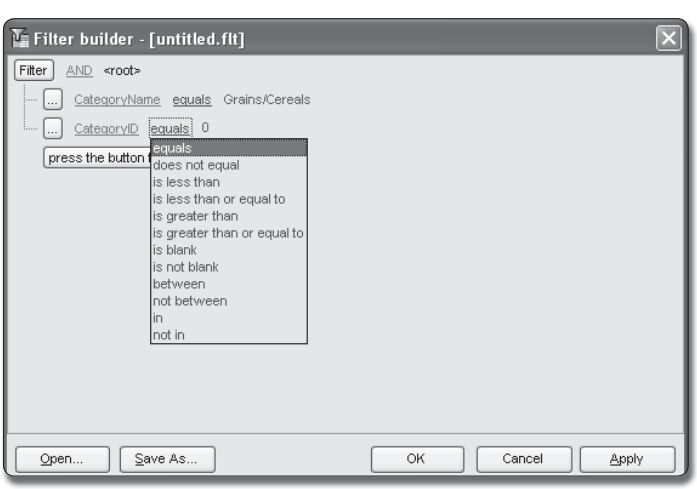


Figura 11. Editor de filtros dos dados

Assim, o mesmo vale para campos de data e alfa-numéricos.

Você também pode salvar os filtros (com a extensão *FLT*) a serem usados em outra oportunidade. Se precisarmos remover os filtros dos dados, basta clicar no X na parte inferior.

Ao clicar com o botão direito no *grid*, temos um menu com várias opções (**Figura 12**), onde veremos as principais a seguir.

A primeira, *Copy Cell*, copia para a área de transferência o valor da célula apenas. A opção *Copy All Records*, copia todos os registros da tabela para a área de transferência, inclusive com o cabeçalho. Faça um teste, escolha a opção e cole em um bloco de notas, por exemplo. Já a opção *Copy Selected Records* copia somente os campos e valores da linha selecionada.

Exportando dados

Através do item de menu *Manipulation*, temos várias opções de exportação, como: *Excel*, *Word*, *TXT* etc. Acesse o item *Manipulation>Export data of "NomedaTabela"* e veja na tela que abre a quantidade de formatos de exportação que temos (**Figura 13**).

Nas próximas telas do *wizard* podemos configurar os campos que serão exportados, bem como o formato dos dados (data, moeda, boleano etc.) e também o formato do arquivo de exportação (para *Excel, HTML* etc.).

O item *Export "NomedaTabela" as SQL Script* do menu *data Manipulation*, exporta os dados da tabela no formato de *script* para execução em outro banco. A vantagem: o formato do *script* pode ser configurado para os seguintes servidores: *DB2, InterBase/Firebird, SQL Server, MySQL, Oracle e PostGreSQL*.

Ou seja, a sintaxe do *script* de exportação, muda de acordo com o servidor escolhido, trazendo a vantagem de exportar dados de um servidor para outro, sem problemas nas diferenças de sintaxe. Veja na **Listagem 1** a diferença do *script* de exportação para alguns servidores.

Veja que temos diferenças significativas de acordo com o servidor, por exemplo, o *SQL Server* coloca "[]" no nome dos campos, enquanto que o *InterBase/Firebird*, usa aspas.

Já o *MySQL* usa apóstrofo e temos apenas um *INSERT* e as linhas dos registros exportados, colocadas após o *VALUE*, onde a última deve ser finalizada com ponto-e-vírgula (;).

Importando dados

No menu *Data Manipulation* temos ainda a opção de importação de dados, onde os mesmos podem vir de formatos como: *Excel, Access, DBF, XML, TXT* etc. para indicar qual campo deve receber determinado valor, tudo dependerá do tipo de formato escolhido.

Se escolher o formato texto, deverá indicar onde começa e onde termina a linha do campo a ser escolhido (**Figura 14**).

Se usarmos, por exemplo, o *Excel*, bastará indicar a célula em que os dados do campo estarão disponíveis. Se o formato for *DBF*, basta escolher o campo da tabela e o campo no arquivo *DBF*.

Filtro rápido

Voltando à parte de filtragens, temos uma opção bastante útil quando queremos marcar um campo e filtrar os dados que sejam iguais, diferentes, maior ou menor que o valor do campo.

Essa opção é melhor em relação ao que mostramos anteriormente, quando a quantidade de registros for muito grande e não vamos escolher o valor na lista de seleção do campo. Basta clicar com o botão direito, escolher o item *Quick Filter* e clicar sobre uma das opções: =, <>, <, >, <=, >= (**Figura 15**).

Com isso, os dados serão filtrados rapidamente. Por exemplo, em uma tabela onde você tenha salvo o código da categoria e quiser filtrar os dados da categoria de código 1, basta clicar no campo de valor igual e escolher a opção = *Value* no item de menu. Rapidamente, os dados serão filtrados onde o código da categoria seja igual a 1.

Editor de campos BLOB

Temos um editor de campos *BLOB*, onde o mesmo está disponível somente quando um campo desse tipo estiver presente na tabela. Para acessá-lo, basta dar um duplo clique no campo e veremos o editor, onde podemos visualizar os dados nas seguintes abas: *Hexadecimal*, *Text*, *Rich Text*, *Image* e *HTML* (**Figura 16**).

Podemos editar o campo para carregar ou salvar imagens, formatar o texto que es-

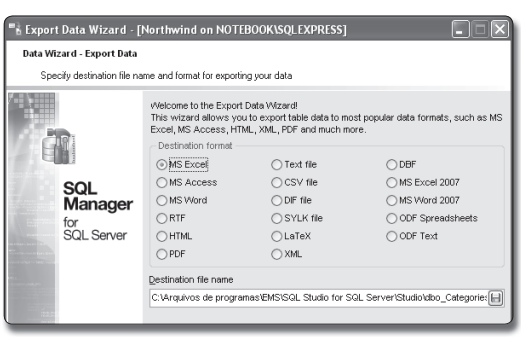


Figura 13. Formatos de exportação de dados

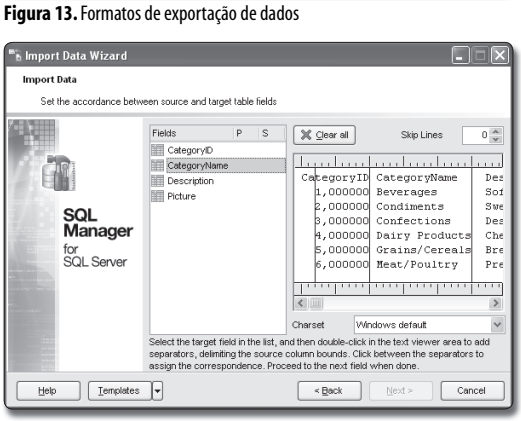


Figura 14. Importando dados de um arquivo texto

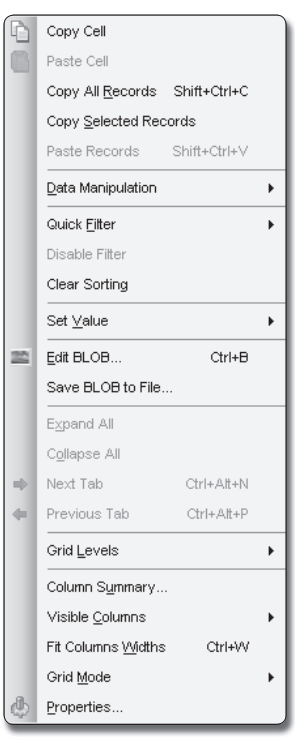


Figura 12. Menu com várias opções para manipular os dados

tiver salvo no campo, através de uma barra de ferramentas que aparece no editor.

Customizando as colunas dos dados

Ainda através do menu de contexto, podemos customizar os campos do *grid*, como mostrar/ocultar campos (item *Visible Columns*) e ainda deixar que os campos fiquem do tamanho do valor salvo nos mesmos, através da opção *Fit Columns Widths*.

Por padrão, os dados carregados no *grid*, chegam à quantidade de 1.000, se quiser carregar todos os dados, basta acessar a opção *Grid Mode>Load All Rows*. Se quiser, que os dados sejam carregados na sua totalidade, acesse o menu *Options>Environment Options*. No item *Grid>Data Options*, marque a opção *Select all records from a table*.

Podemos adicionar somatórios no rodapé do *grid* que mostra os dados. Acesse o item *Column Summary* e no editor (**Figura 17**), escolha os campos que deseja mostrar um sumário, onde podemos escolher o tipo (somatório, contador, média etc).

Os valores podem ser escolhidos em nível de campo, onde também podemos configurar o cabeçalho e rodapé de um grupo, por exemplo, e também o formato que teremos para o campo. Veja na **Figura 18**, que colocamos um contador no cabeçalho (campo *CategoryID*) e somatório no campo *UnitPrice* (rodapé) da tabela *Products*.

Voltando a aba *Data* na parte inferior, podemos visualizar os dados no formato de *Grid* (*Grid View*), formulário (*Form View*), relatório (*Print Data*) e editor de *BLOB*, quando for o caso.

Editor de SQL

A ferramenta possui um ótimo editor para criar consultas *SQL*. E o mais interessante, que todas as opções mostradas até aqui, estão presentes no *grid* que mostra os dados da consulta, ou seja, você pode filtrar, exportar, customizar colunas.

O editor ainda conta com *IntelliSense*, onde podemos apertar a combinação de teclar *CRTL+SPACE* e teremos os objetos do banco de dados (**Figura 19**).

Ainda no editor, podemos carregar arquivos *SQL*, salvar consultas criadas, procurar texto, colocar marcadores (*bookmarks*). Além disso, no editor podemos criar várias consultas, uma em cada aba, assim podemos ter o editor aberto com várias consultas, cada uma sendo executada separadamente.

Também podemos formatar a consulta *SQL*, em um formato pré-definido pela

Listagem 1. Diferenças no script de acordo com o servidor de dados

```
SQL Server
INSERT INTO [dbo].[Categories] ([CategoryID], [CategoryName], [Description])
VALUES (1, N'Beverages', N'Soft drinks, coffees, teas, beers, and ales')
GO

InterBase/Firebird
INSERT INTO "Categories" ("CategoryID", "CategoryName", "Description")
VALUES (1, 'Beverages', 'Soft drinks, coffees, teas, beers, and ales');

MySQL
INSERT INTO `Categories` (`CategoryID`, `CategoryName`, `Description`, `Picture`)
VALUES (1, 'Beverages', 'Soft drinks, coffees, teas, beers, and ales'),
       (2, 'Condiments', 'Sweet and savory sauces, relishes, spreads, and seasonings'),
```

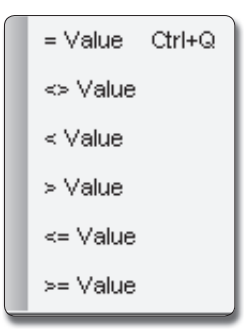


Figura 15. Opções de filtragem rápida

Figura 16. Editor de campos Blob com os formatos de visualização

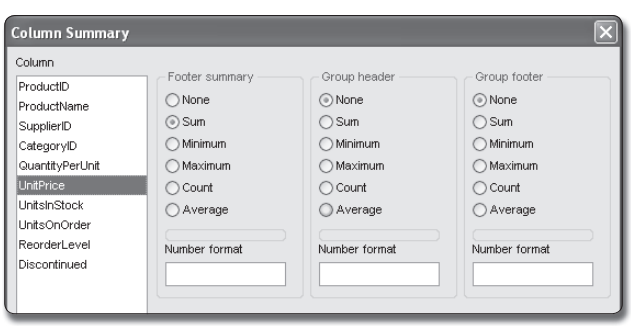


Figura 17. Sumários

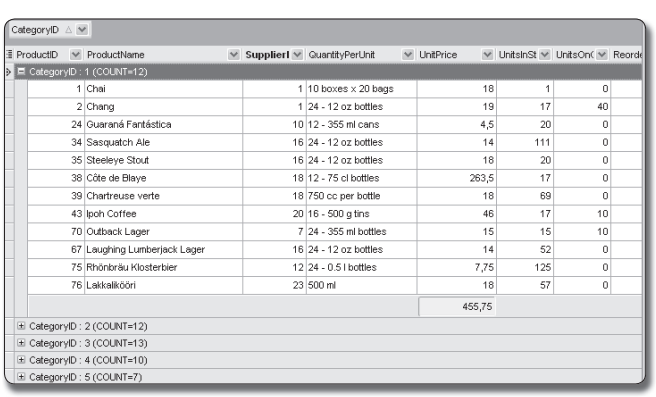


Figura 18. Somatórios por grupo e campo

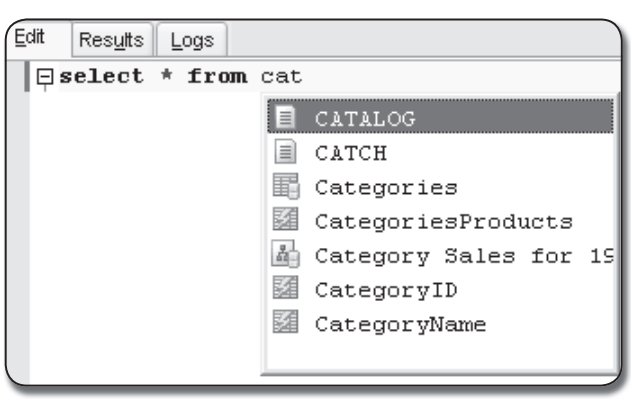


Figura 19. Editor de consultas SQL

ferramenta, assim, depois de digitar um comando *SQL* muito grande, com vários campos, basta teclar *SHIFT+CRTL+F* e o comando será formatado com os campos endentados, assim como as cláusulas *SELECT, FROM, WHERE, INNER* etc.

Construtor de consultas

O editor de consultas é bastante produtivo, mas se a consulta a ser criada, depender de várias tabelas, campos, *views*, temos um construtor de consultas visual, onde escolhemos as tabelas, fazemos o relacionamento entre as mesmas, escolhemos os campos etc.

Acesse o menu *Tools>Show Query Builder* e no lado esquerdo, temos um painel com as tabelas e visões do banco de dados. Basta escolher o objeto e arrastar para o painel cinza.

Para fazer o relacionamento entre as tabelas, basta escolher o campo e arrastar para o campo que será o relacionamento na outra tabela (**Figura 20**). Após, escolha os campos e aperte *F9*. Pronto, sua consulta será mostrada, e ainda pode ser transformada em uma *view*.

Na parte inferior, temos opções para criar critérios, grupos e ordenação dos campos. O *grid* do resultado da consulta possui as mesmas funcionalidades mostradas anteriormente.

Extraindo a definição do banco e dados

Podemos extrair toda a definição do banco de dados com seus respectivos objetos, assim como os dados das tabelas. Para isso, acesse o menu *Tools>Extract Database* e na primeira tela do *wizard* temos que indicar o servidor e o banco de dados (**Figura 21**).

Na próxima tela, podemos abrir o comando no editor ou salvar em um arquivo (com a extensão *SQL*). Caso na primeira tela seja marcada a opção para extrair todos os dados, as próximas telas podem ser diferentes. Não marque essa opção se deseja escolher os objetos e dados a serem extraídos.

Na tela seguinte, temos as opções de exportação de campos *BLOB*. Após, temos as opções do *script* onde podemos configurá-lo para ter o comando de criação do banco. Podemos também configurar, em que quantidade de registros será efetuado o *commit* nos dados. Na próxima tela, finalizamos a extração dos dados.

Documentação do banco de dados

Como frisei bastante no decorrer do artigo, podemos sempre colocar uma descrição para os objetos do banco. Com essa descrição, é muito fácil gerar uma documentação em formato *HTML*. Acesse o menu *Tools>HTML Report* e no assistente, escolha o banco e o diretório onde será gerado o arquivo. Na próxima tela (**Figura 22**), escolha os objetos que deseja gerar a documentação.

Nas próximas telas, podemos configurar o *CSS* (arquivo de estilo) do *HTML* a ser gerado, bem como o cabeçalho e rodapé. Ao final do assistente, podemos abrir o relatório e visualizar o resultado (**Figura 23**).

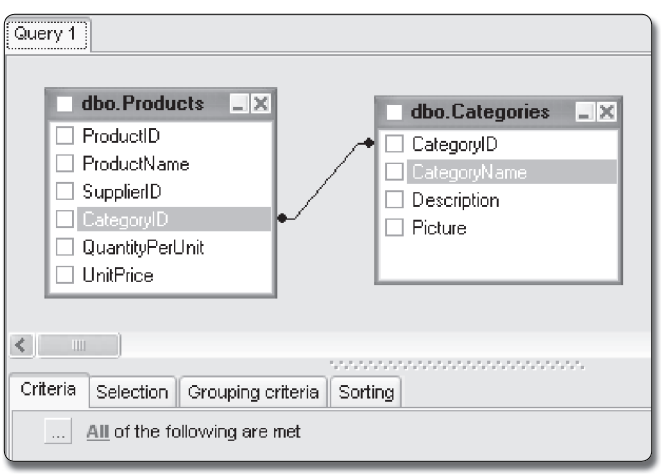


Figura 20. Criando consultas com o editor visual

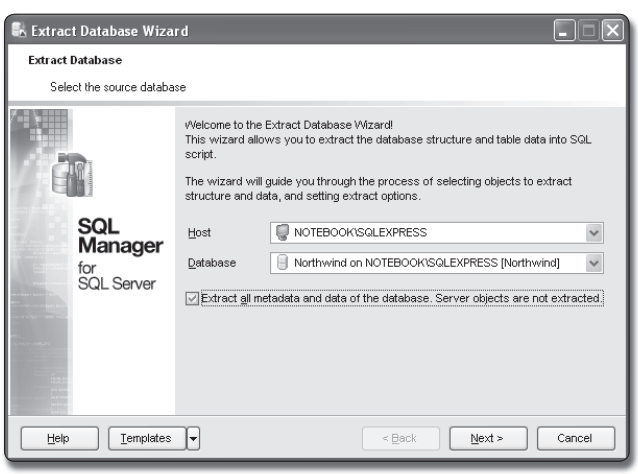


Figura 21. Extrator de objetos e dados do banco

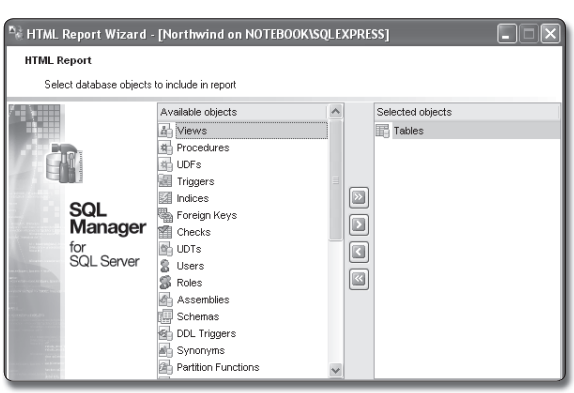


Figura 22. Escolhendo os objetos para a documentação

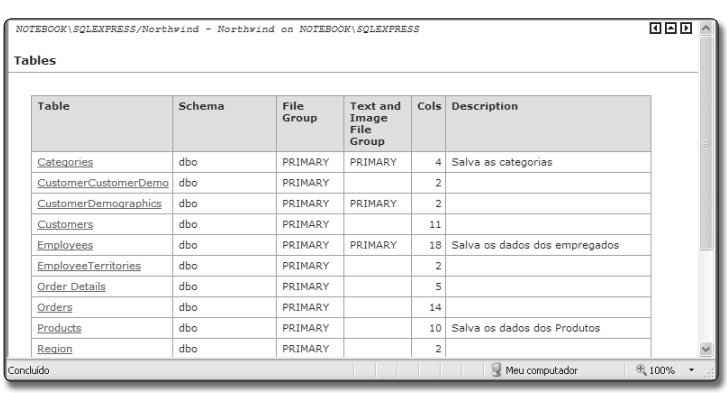


Figura 23. Relatório com a descrição dos objetos do banco

Outra maneira de fazer uma documentação do banco é através dos relatórios que a ferramenta gera automaticamente. Acesse o menu *Tools>PrintMetadata* e escolha os objetos no painel que será mostrado.

Após, clique em *Preview* e veja um relatório (**Figura 24**) com as características do objeto (campos, instruções etc). O gerador de relatórios da ferramenta exporta os relatórios para os seguintes formatos: *HTML, Excel, Texto, RTF, CSV, BMP, JPEG e TIFF*.

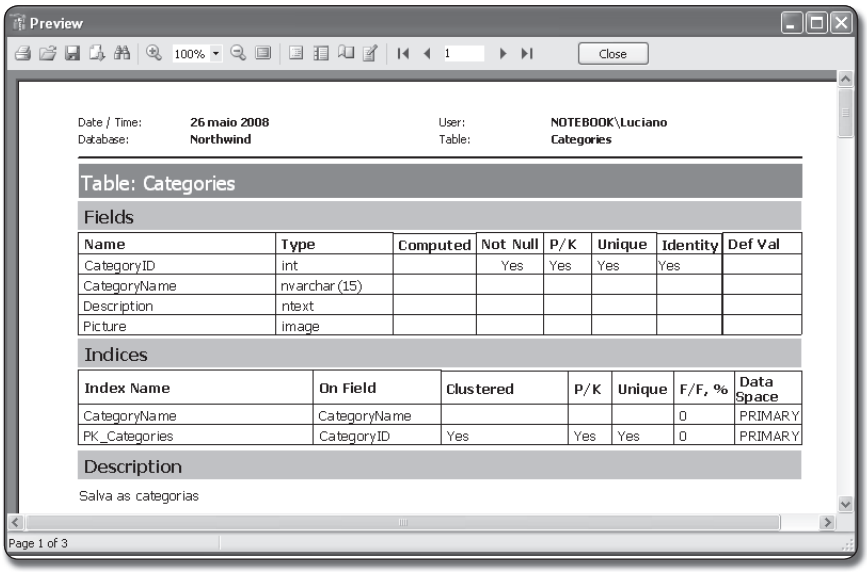


Figura 24. Gerando relatórios dos objetos do banco

Usuários e permissões

O *SQL Manager* possui um controle de usuários e permissões bastante amplo e muito fácil de usar. Por exemplo, para trabalhar com *logins*, acesse o menu *Tools>Login Manager*.

Temos os *logins* criados e podemos adicionar/editar/excluir e exportar a lista de *logins*. Para o *login* temos todas as permissões do banco, servidor etc., para configurar. Para trabalhar com permissões, acesse o menu *Tool>Grant Manager*.

No editor (**Figura 25**), podemos filtrar os objetos e executar as permissões de cada um, apenas com um duplo clique no mesmo ou clicando com o botão direito, temos um menu de contexto com todas as opções (*Grant*, *Denny* etc.).

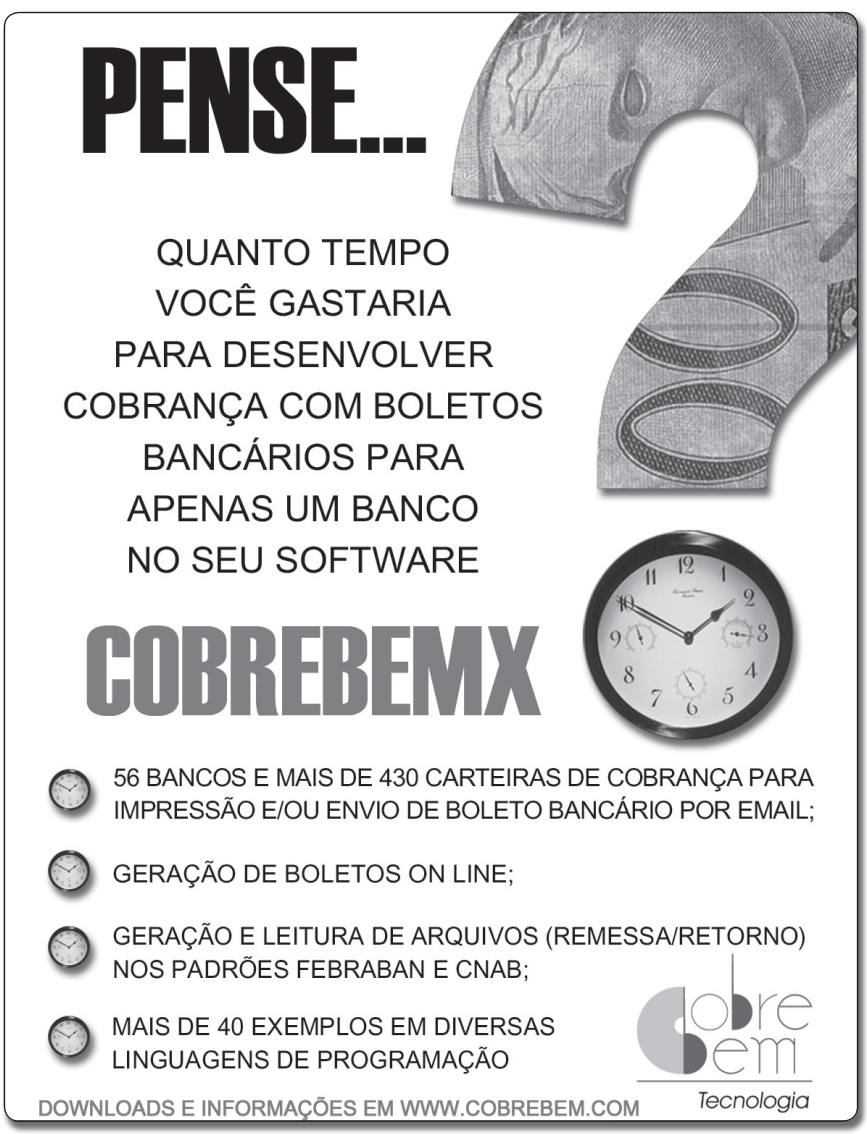
Fácil, rápido e simples. No caso de tabelas e *views*, podemos ainda, dar permissão em nível de campo, para *Select*, *Update* e *References*.

Visual Database Designer

Que tal, começarmos a desenhar em vez de criar o nosso banco de dados? Como assim? Em vez de começarmos no editor de *SQL* a digitar código, podemos desenhar as tabelas e depois gerar o banco. O que acham? Mais fácil e produtivo.

Acesse o menu *Tools>Visual Database Designer* e será aberto um *designer* para criação dos objetos do banco. Clique com o botão direito no editor e escolha *Create>Create Table* para abrirmos um editor onde vamos criar os campos da tabela (igual ao editor da ferramenta para criação de tabelas).

Após criar a tabela a mesma será mostrada no *designer*. Assim, basta criar



Listagem 2. Comandos SQL executados no banco

```
[26/5/2008 18:02:22.562]
Operation : SELECT @@SPID
[26/5/2008 18:02:22.656 (00:00:00,094)]
Result : "OK"
[26/5/2008 18:02:22.671]
Operation : SELECT spid, CASE WHEN (spid = @@SPID)
  THEN 1 ELSE 0 END isself FROM master..sysprocesses
   WHERE hostprocess = 1644
[26/5/2008 18:02:22.906 (00:00:00,235)]
Result : "OK"
[26/5/2008 18:02:22.937]
Operation : select * from categories
StatementId = 1
...
[26/5/2008 18:02:58.828]
Operation : CREATE TABLE [dbo].[TESTE] (
  [ID] int IDENTITY(1, 1) NOT NULL,
  [TESTE] nvarchar(max) NULL,
  PRIMARY KEY CLUSTERED ([ID])
)
[26/5/2008 18:02:58.843 (00:00:00,015)]
Result : "OK"
...
[26/5/2008 18:03:38.109]
Operation : DROP TABLE [dbo].[TESTE]
[26/5/2008 18:03:38.109 (00:00:00,000)]
Result : "OK"
```

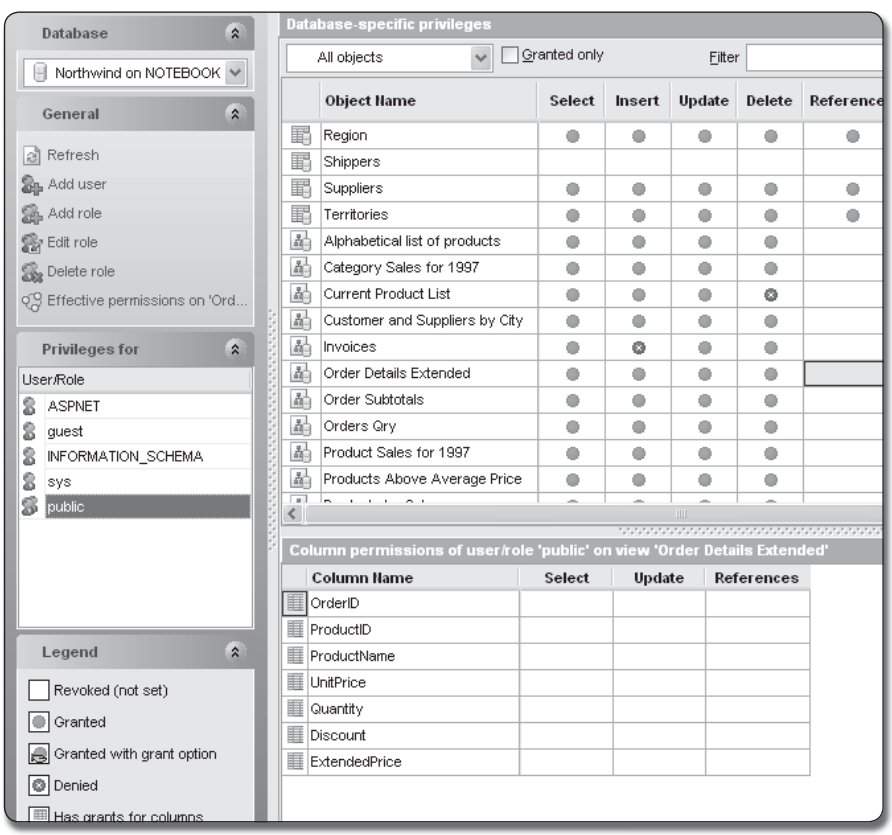


Figura 25. Editor de permissões

a próxima tabela. Para relacionar as tabelas, basta clicar de direita e escolher *Create>Create Relation*. O editor será o mesmo da **Figura 8** e basta escolher os campos relacionados.

Após, você possui um diagrama que pode ser salvo, exportado como imagem (*BMP*, *JPEG*, *GIF* ou *WMF*) ou impresso. Podemos ainda fazer engenharia reversa de banco de dados, ou seja, podemos gerar o diagrama baseado em um banco de dados pronto (**Figura 26**).

Monitor SQL

Se você precisa monitorar o que está sendo executado no banco (consultas, *Stored Procedures*, criação/exclusão de tabelas etc.), basta acessar o menu *Tools>SQL Monitor*. Deixe a janela aberta e faça consultas, crie e exclua tabelas, execute *Stored Procedures* etc.

Volte novamente ao *SQL Monitor* e veja o *log* de comandos executados. Veja na **Listagem 2** os principais comandos que são mostrados no *SQL Monitor*.

Serviços

O menu *Services* concentra as funcionalidades para *backup/restore* e características comuns ao servidor escolhido. Por exemplo, para o *SQL Studio for InterBase/Firebird*, não temos as opções *Attach Database* e *Detach Database* presente no *SQL Studio form SQL Server*.

Conclusão

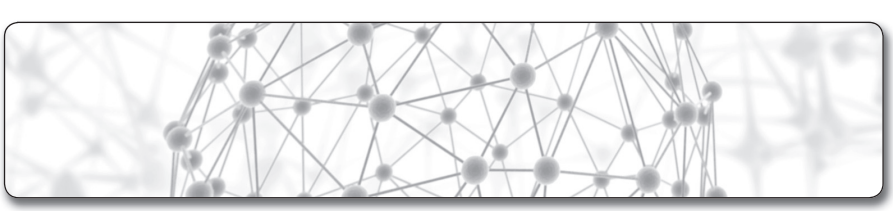
Vimos neste artigo como podemos trabalhar com vários servidores de banco de dados, usando uma ferramenta de características e funcionalidades iguais, sem precisarmos aprender como a ferramenta funciona, de acordo com o banco de dados. Um grande abraço a todos! ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



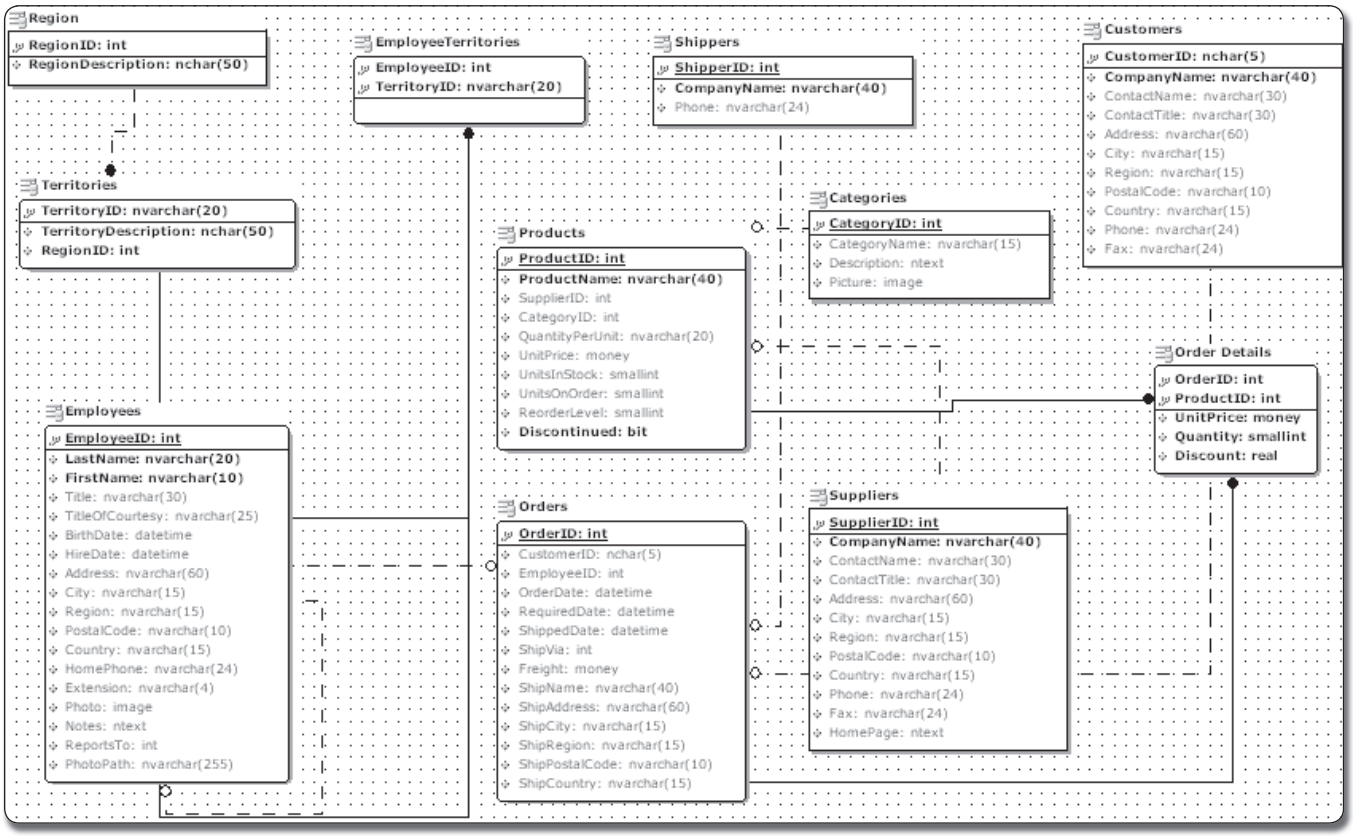


Figura 26. Diagrama com banco de dados pronto

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Globalizando Aplicações

Traduzindo aplicações em ASP.NET

Resumo DevMan

Entre os diversos recursos do ASP.NET 2.0 está o Globalization, um recurso bastante útil e prático quando se pensa em desenvolver uma aplicação que permita sua visualização em diversas linguagens. Faremos o desenvolvimento de um web site cujo o principal objetivo é fazer com que o site detecte a localização do internauta, ou seja, sua linguagem e redirecione as páginas para a linguagem adequada. Veremos como esse recurso é transparente e simples de ser aplicado em qualquer web site ASP.NET.

Nesse artigo veremos

- Novidades do ASP.NET 2.0;
- Globalization;
- Técnicas para tradução de sites em ASP.NET.

Qual a finalidade

- Com os recursos do ASP.NET 2.0 é possível preparar a página para se comportar de maneiras diferentes de acordo com a localização do internauta.

Quais situações utilizam esses recursos?

- A principal aplicabilidade desses recursos diz respeito a possibilidade de manter um mesmo site em diversas linguagens sem a necessidade de muito trabalho de desenvolvimento.

Manoel Edesio B da Silva

manoel.edesio@hkconsultoria.com.br

detém as certificações em Borland C++ Builder, Borland Delphi for Win32 e Borland Delphi for .NET. Palestrou para a Borland em Delphi Meetings em alguns estados brasileiros, palestrante na Borland Conference de 2004, 2005, 2006 e 2007. Atualmente é diretor da HK Consultoria, empresa especializada em consultoria, treinamentos customizados e desenvolvimento de software em produtos Borland. Além dos quesitos anteriores a HK atua com projetos especializados em mobilidade com utilização de celulares e PDA'S, além de ser parceira PDA da operadora Claro.

Sempre ao navegarmos na internet, reparamos sites que dão a possibilidade do usuário visualizá-lo em diferentes linguagens, porém esse recurso é sempre visto nos sites e aplicações através de botões e imagens que representam as traduções disponíveis, "as típicas bandeirinhas". Ao clicar em uma delas o site/aplicação é, geralmente, redirecionado para a página específica daquela linguagem. Porém, para manter uma aplicação dessas temos um custo muito elevado com manutenção, visto que precisamos manter diferentes cópias da mesma aplicação em cada idioma que se deseja disponibilizar.

Nesse artigo vamos tratar de duas técnicas chamadas de *Globalization* e *Localization* existentes no .NET Framework que facilitam o uso para implementação desse tipo de recurso. Esse é um recurso pouco comentando e pouco visto em livros técnicos da área, mas vamos trazer aqui a implementa-

ção básica e avançada desse recurso.

Globalization e Localization, o que são?

Globalization é o fato de tornarmos a nossa aplicação de forma compatível a diferentes tipos de usuário que podemos classificar como usuários de países diferentes ou que utilizam em determinado país um idioma diferente como preferência. Porém, não podemos considerar a nossa aplicação globalizada apenas pelo fato de traduzir a mesma para diversas línguas. Esse tipo de trabalho parece ser ágil em curto prazo, mas nos dará posteriormente muitas dores de cabeça com manutenção da aplicação. Além de que, a tradução é o meio principal da globalização e mais conhecido, porém além da tradução precisamos adaptar a formatação dos valores, datas etc. Isso é exatamente o papel do *Localization*. Esse recurso tem a funcionalidade de adaptar a nossa aplicação de acordo com as preferências culturais do usuário.

Identificando a preferência do usuário

Para que a aplicação seja globalizada automaticamente, primeiramente temos que identificar através do *request* do usuário qual é a cultura dele. A cultura é selecionada através do browser do usuário. No caso do *Internet Explorer* podemos conferir a preferência acessando o menu *Ferramentas>Opções da Internet>Idiomas* e teremos as configurações conforme **Figura 1.**

Criando a primeira aplicação

Para entendermos melhor como funciona esse processo, vamos colocar a mão na massa e criar uma pequena aplicação. Abra o RAD Studio 2007 e crie uma nova aplicação utilizando o menu *File|New>Web Application – Delphi for .NET*. Salve o projeto com o nome de *Globalization_01.dproj*. Mantenha o nome da página principal como *default.aspx*. Nesse exemplo, vamos imaginar que temos uma aplicação que o padrão de idioma é o *Português*, mas queremos que ela tenha uma versão em *Inglês* e outra em *Espanhol*. Para o primeiro exemplo, teremos páginas duplicadas e com os textos traduzidos para os formatos necessários. Com isso, ao usuário acessar a aplicação, de acordo com a cultura escolhida em seu *browser*, ele será redirecionado para a página correspondente.

Adicione a nossa aplicação mais uma página através do menu *File|New>Other>Delphi for .NET Projects>New ASP.NET Files>ASP.NET Page* com o nome *ingles.aspx*. Repita a operação e adicione uma nova página, desta vez com o nome *espanhol.aspx*.

Configure a página *default.aspx* como página inicial clicando com o botão direito e escolhendo a opção *Set as Start Page*. Para diferenciarmos cada página, vamos adicionar um *Label* ao centro e trocar a propriedade *Text* em cada para *Português*, *Inglês* e *Espanhol*, respectivamente.

Após essa etapa, vamos acessar a nossa página *default.aspx* e dar um duplo clique no *WebForm* para acessar o evento *Page_Load* da página. Digite o código da **Listagem 1.**

No início do código, através da classe *request*, temos a propriedade *UserLanguages*. Nela, estamos pegando o conteúdo do item *0* correspondente à preferência de linguagem do usuário e em seguida os dois primeiros caracteres que correspondem por exemplo ao *pt* de *Português*. Em seguida identificamos qual a preferência escolhida e direcionamos o mesmo para a página correspondente ao idioma.

Para testar, execute a aplicação e note que será direcionado automaticamente para a preferência de idioma configurada em seu *browser*. Para testar as outras opções, acesse *Ferramentas|Opções de internet>Geral>Idiomas*. Clique em *Adicionar* e escolha as linguagens que preferir. Depois mova-as para cima de acordo com a sua preferência. Execute novamente a aplicação e perceba que a cada vez que mudamos a nossa preferência a página é redirecionada para o idioma correto.

Utilizando recursos profissionais

No exemplo anterior, conseguimos demonstrar que, de acordo com a cultura escolhida, podemos redirecionar o usuário para cada página equivalente, porém nesse caso precisamos ter três páginas que apenas mudam o texto para cada língua. Essa técnica, além de ser muito trabalhosa, não torna a nossa aplicação profissional, visto que a cada novo componente adicionado teremos que replicar para as outras, além, é claro, de ter o esforço de traduzir o texto. Imagina quando resolvemos então adicionar páginas novas em nossa aplicação? Isso gera sempre um enorme trabalho de manutenção.

O que podemos fazer para resolver esse problema é utilizar um recurso do .NET Framework chamado *Satellite Assemblies*, onde conseguimos, através de uma única aplicação, ter todos os componentes traduzidos para cada língua específica sem muito esforço.

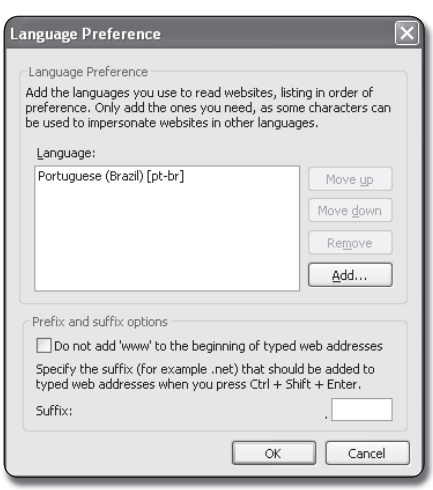


Figura 1. Configuração das preferências de Idioma

Listagem 1. Implementando o evento Page_Load do default.aspx

```
procedure TDefault.Page_Load(sender: System.Object;
  e: System.EventArgs);
var
  Lingua: string;
begin
  Lingua := Request.UserLanguages[0].Substring(0, 2).
    ToLower();
  if Lingua = 'es' then
     Response.Redirect('espanhol.aspx')
  else if Lingua = 'en' then
    Response.Redirect('ingles.aspx');
end;
```

A premissa básica é que todos os componentes em tela sejam controles *Server Controls*, independente de ser um *HtmlControl* ou um *WebControl*, pois eles serão primeiramente carregados no servidor para depois serem carregados em nosso navegador através de um recurso chamado *ResourceManager*. Os textos traduzidos para cada controle ficarão no que chamamos de *Satellite Assemblies* que não passa de um arquivo XML com extensão *.resx*. Vamos editar esse arquivo no próprio RAD Studio 2007. Esse arquivo XML conterá as informações para cada controle, onde identificará o nome do controle e o valor dele associado.

Vamos então colocar a mão na massa novamente e criar um novo projeto onde teremos tradução baseado no arquivo XML de configurações. No *Rad Studio 2007* crie uma nova aplicação usando *File|New|ASP.NET Web Application – Delphi for .NET*. O nome da aplicação será *Globalization_02.dproj* e *Default.aspx* renomeie para *Login.aspx*.

Vamos desenhar a interface semelhante à **Figura 2.** Adicione dois *Label's*(*lblMensagem1* e *lblMensagem2*) na parte superior aumentando sua fonte (*Size*) para *Medium* e *Name* para *Verdana*. Troque seu *Caption* para *Autenticação Requerida*. Em um segundo *Label* apenas modifique a propriedade *Text* para "Para ter acesso é necessária autenticação. Digite usuário e senha.". Adicione uma tabela de duas colunas e duas linhas. Inclua em cada célula da segunda coluna um *TextBox* com os nomes *edtUsuario* e *edtSenha*, respectivamente. Ao lado de cada *TextBox* insira *Label's*(*lblUsuario* e *lblSenha*) para identificar o tipo de informação. Para finalizar adicione dois *Button's* e troque os nomes para *btnLogar* e *btnCancelar*.

Agora precisamos criar no RAD Studio quatro arquivos XML para guardarmos as informações de tradução. Então, acesse o menu File|New>Other>Delphi for .NET Projects>New Files>Resource File. Renomeie o arquivo com o nome strings.pt.resx. Esse arquivo será o que vamos adicionar as configurações para o idioma Português. Repita o processo para incluirmos os outros dois arquivos, ou seja, um para cada idioma. Utilize os nomes strings.en.resx para Inglês e strings.es.resx para Espanhol. Um quarto arquivo deverá ser adicionado, desta vez com o nome strings.resx. Esse será o idioma default, servirá para quando o idioma escolhido no browser não for suportado pelos Satellites Assemblies de nossa aplicação.

Feito isso vamos adicionar no arquivo strings.pt.resx logo após o fechamento da tag </resheader> do item <resheader name="Writer"> o código para tradução dos componentes da página login.aspx conforme Listagem 2.

Vamos entender o código: repare que temos a tag <data> que identifica o nome do componente dentro de uma determinada página e logo abaixo temos a tag <value> que informa qual é o conteúdo que vamos colocar na tag. Repita esses passos para os arquivos strings.en.resx, strings.es.resx e strings.resx que receberão os códigos das **Listagens 3, 4 e 5**, respectivamente.

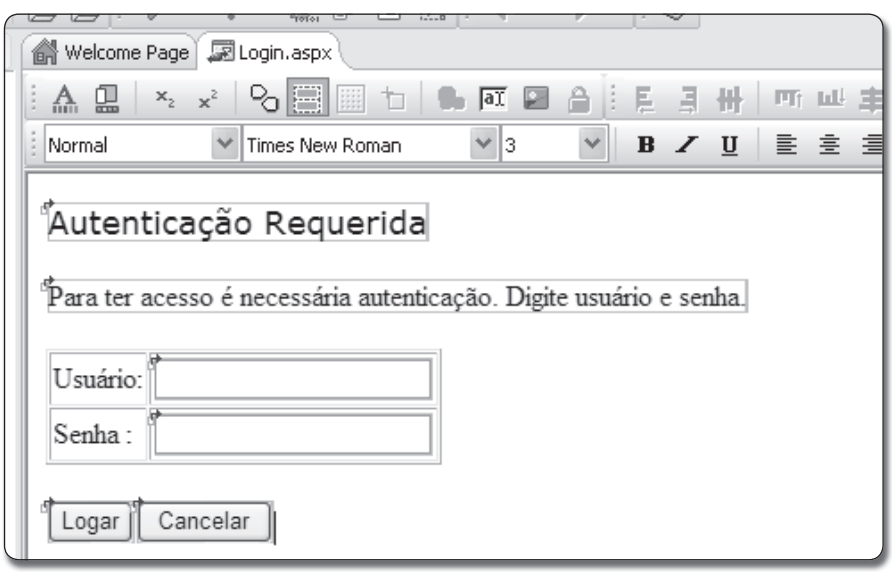


Figura 2. Exemplo de janela de login

Listagem 2. Código do arquivo strings.pt.resx para tradução da página login.aspx

```
<data name="login.lblTexto01">
  <value>Autenticação Requerida</value>
</data>

<data name="login.lblTexto02">
  <value>Para ter acesso a aplicação, e
    necessário autenticacao. Informe usuário
    e senha para acesso.</value>
</data>

<data name="login.lblUsuario">
  <value>Usuário</value>
</data>

<data name="login.LBLSenha">
  <value>Senha</value>
</data>

<data name="login.btnLogar">
  <value>Logar</value>
</data>

<data name="login.btnCancelar">
  <value>Cancelar</value>
</data>
```

Nota do DevMan

Server Controls são controles *ASP.NET* que rodam no servidor, como por exemplo, botões, *textbox's, listbox's* etc. *HTML Controls* também podem rodar no servidor e serem manipulados da mesma forma que Server Controls, bastando para isso adicionar a tag *runat="Server"* a sua declaração.

Listagem 3. Código do arquivo strings.en.resx

```
<data name="login.lblMensagem1">
  <value>Required authentication</value>
</data>
<data name="login.lblMensagem2">
  <value>To have access the application,
    is necessary authentication. Input the
    user and password for access.
  </value>
</data>
<data name="login.lblUsuario">
  <value>User</value>
</data>
<data name="login.lblSenha">
  <value>Password</value>
</data>
<data name="login.btnLogar">
  <value>Login</value>
</data>
<data name="login.btnCancelar">
  <value>Cancel</value>
</data>
```

Listagem 4. Código do arquivo strings.es.resx

```
<data name="login.lblMensagem1">
  <value>Autentificacion requerida</value>
</data>
<data name="login.lblMensagem2">
 <value>Tener acceso el uso, es
    autentificacion necesaria. Entre el
    usuario y la contrasena para el
    acceso
  </value>
</data>
<data name="login.lblUsuario">
  <value>Usuario</value>
</data>
<data name="login.lblSenha">
  <value>Contrasena</value>
</data>
<data name="login.btnLogar">
  <value>Conexion</value>
</data>
<data name="login.btnCancelar">
  <value>Cancelacion</value>
</data>
```

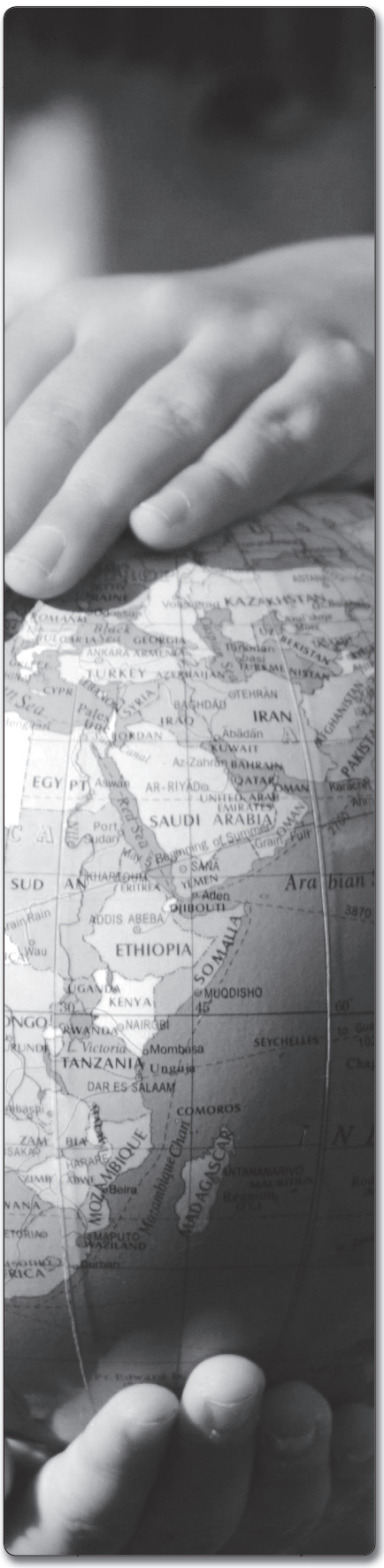
Listagem 5. Código do arquivo strings.resx

```
<data name="login.lblTexto01">
  <value>Autenticacao Requerida (default)</value>
</data>
<data name="login.lblTexto02">
  <value>Para ter acesso a aplicacao, e
    necessario autenticacao. Informe usuario
    e senha para acesso.</value>
</data>
<data name="login.lblUsuario">
  <value>Usuario</value>
</data>
<data name="login.LBLSenha">
  <value>Senha</value>
</data>
<data name="login.btnLogar">
  <value>Logar</value>
</data>
<data name="login.btnCancelar">
  <value>Cancelar</value>
</data>
```

Listagem 6. Implementando o evento Page_Load

```
procedure TDefault.Page_Load(sender: System.Object;
  e: System.EventArgs);
var
  FCulturaUsuario: string;
  FobjStringsTrad: ResourceManager;
begin
  FobjStringsTrad := ResourceManager.Create('strings',
    Assembly.GetExecutingAssembly);
  FCulturaUsuario := HttpContext.Current.Request.
    UserLanguages[0];
  Thread.CurrentThread.CurrentCulture :=
    CultureInfo.CreateSpecificCulture(FCulturaUsuario);
  Thread.CurrentThread.CurrentUICulture :=
    CultureInfo.Create(FCulturaUsuario);

  lblMensagem1.Text := FobjStringsTrad.GetString('login.lblMensagem1');
end;
```



Definidos os quatro arquivos *.resx* com as traduções, vamos implementar um pequeno trecho de código para ver como funciona. No formulário *login.aspx* dê um duplo clique na página para acessar o evento *Page_Load* e adicione o código da Listagem 6, que faz a tradução de um componente. Veja na **Figura 3**

que o primeiro *Label* ("lblMensagem1") apresenta a mensagem *Required authentication*, ou seja, uma mensagem em inglês. O que significa que o *Label* foi traduzido com base no arquivo *strings.en.resx*.

Listagem 7. Declaração da classe TTranslate

```
...
type
  TTranslate = class
  private
    FCulturaUsuario: string;
    FobjStringsTrad: ResourceManager;
    FWebFormName: string;
    FileOutput: TextFile;
  public
    constructor Create;
    procedure InicializeTranslate(vtranslate: string);
    procedure TranslateControl(fcontrol: Control);
    procedure SearchControl(fcontrol: Control);
    property WebFormName: string read FWebFormName
      write FWebFormName;
    property CulturaUsuario: string read
      FCulturaUsuario write FCulturaUsuario;
    Property objStringsTrad: ResourceManager read
      FobjStringsTrad write FobjStringsTrad;
  end;
...
```

Listagem 8. Implementação das funções e procedimentos

```
implementation
{ TTranslate }
constructor TTranslate.Create;
begin
  inherited;
end;
procedure TTranslate.InicializeTranslate(vtranslate:
  string);
begin
  WebFormName := Copy(vtranslate, 1, Pos('_', vtranslate) - 1);
  objStringsTrad := ResourceManager.Create('strings', Assembly.GetExecutingAssembly);
  CulturaUsuario := HttpContext.Current.Request.UserLanguages[0];
  Thread.CurrentThread.CurrentCulture := CultureInfo.CreateSpecificCulture(CulturaUsuario);
  Thread.CurrentThread.CurrentUICulture := CultureInfo.Create(CulturaUsuario)
end;
procedure TTranslate.SearchControl(fcontrol: Control);
var
  MyControl: control;
  MyText: string;
  i: integer;
begin
  i := 0;
  for MyControl in (fcontrol).Controls do
  begin
    if (fcontrol.Controls[i] is &Label) then
    begin
      MyText := Self.objStringsTrad.GetString(Self.WebFormName + '.' + (fcontrol.
Controls[i] as &Label).ID);
      if not (MyText = '') then
        (fcontrol.Controls[i] as &Label).text := MyText;
    end;
    if (fcontrol.Controls[i] is Button) then
    begin
      MyText := Self.objStringsTrad.GetString(Self.WebFormName + '.' + (fcontrol.
Controls[i] as Button).ID);
      if not (MyText = '') then
        (fcontrol.Controls[i] as Button).text := MyText;
    end;
    if (fcontrol.Controls[i] is CheckBox) then
    begin
      MyText := Self.objStringsTrad.GetString(Self.WebFormName + '.' +
        (fcontrol.Controls[i] as CheckBox).ID);
      if not (MyText = '') then
        (fcontrol.Controls[i] as CheckBox).text := MyText;
    end;
    inc(i);
  end;
end;
procedure TTranslate.TranslateControl(fcontrol: Control);
begin
  SearchControl(fControl);
end;
```

Automatizando a tradução

Primeiramente vamos entender o código implementado na página. Na primeira linha da Listagem 6 carregamos o arquivo de recursos baseado no nome *strings* para em seguida utilizá-lo na tradução dos itens. Na segunda linha buscamos o idioma de preferência do usuário e em seguida, na terceira linha, informamos ao .NET Framework qual cultura utilizar quando precisar exibir informações como *data*, *valores* etc. Na quarta linha estamos informando ao *Framework* que temos uma cultura específica e não uma cultura neutra (a *default*). Em seguida utilizamos o método *GetString* para capturar o valor na língua específica do controle. Porém reparem que ao implementar esse código para cada controle de nossa aplicação não será nada fácil, imaginem. Perceba que precisamos programar o *Label* ("lblMensagem1") para receber o resultado de *GetStrings*. Realmente seria uma tarefa super tediosa.

Para resolver tal problema, vamos utilizar algumas técnicas de programação POO para aperfeiçoar todo o código do nosso projeto. Então adicione uma nova *Unit* ao projeto através do menu *File|New>Other>Delphi for .NET Projects>New Files>Unit*. Em seguida pressione *CTRL + S* e salvar a *Unit* com o nome de "uTranslator.pas".

Localize a palavra reservada *Interface* e abaixo dela declare uma seção *Uses* declarando as seguintes namespaces: *Uses System.Globalization, System.Threading, System.Resources, System.Reflection, System.Web, System.Web.UI* e *System.Web.UI.WebControls*.

Agora declare uma classe *TTranslate* como na **Listagem 7** e após a definição pressione as teclas *Shift + Ctrl + C* para gerar os cabeçalhos dos métodos. Gerados os respectivos cabeçalhos dos métodos, codifique-os de acordo com a **Listagem 8.**

Vamos entender o código da **Listagem 8.** No método *InicializeTranslate*, estamos inicializando a configuração de preferência de idioma e os *Threads* de *Cultura*. Na verdade, estamos portando o código do evento *Page_Load* da página para esse método visando facilitar a implementação. Esse método receberá como parâmetro o nome da classe do *WebForm*. O Método *TranslateControl* apenas faz a chamada para o *SearchControl* passando como parâmetro o controle onde deve-

mos pesquisar por componentes.

Já o *SearchControl*, esse sim que fará todo o trabalho duro. Ele recebe como parâmetro o controle, onde será passado um *HtmlForm*, que é a página container que contém todos os controles. Em seguida, temos um *for* no *array* de controles desse elemento onde se verifica via *TypeCast* se ele é do tipo *Label*. Em caso positivo utilizamos o método *GetString*, como já vimos antes, só que de forma dinâmica, ou seja, pegamos nome do *WebForm* e o *ID* do controle para montar a *string* de busca. Havendo um conteúdo para ser traduzido, devolve-se o mesmo para o controle atual. Por fim repetimos todo o processo desta vez fazendo o *TypeCast* com outros tipos de controles.

Para completar essa otimização e verificar o exemplo funcionando, vamos acessar o evento *Page_Load* da página *login.aspx* e alterar o código que tínhamos colocado anteriormente, para o código da **Listagem 9**.

> **Nota**: Não esqueça de pressionar Alt + F11 e adicionar uma referência a *Unit uTranslate* na *Uses* da *Unit login.aspx* para que ela seja capaz de enxergar a classe criada na unit *uTranslate.pas*.

Nesse código estamos usando o método *InicializeTranslate* para efetuar a inicialização e em seguida fazemos um *for* para pegar o item que represente o *HtmlForm*. Passamos o mesmo como parâmetro para o método *TranslateControl* de nossa classe. Esse item é o que contém todos os controles adicionados dentro do nosso *WebForm*. Em seguida, rode a aplicação e modifique a linguagem do seu *browser* para ver os diferentes resultados (**Figuras 4, 5 e 6**).

Conclusão

Podemos perceber por meio desse artigo sobre *Globalization*, que apesar da tradução de aplicações web para os programadores ser algo em princípio bem tedioso, conseguimos através de técnicas presentes no .NET Framework, otimizar todo o processo e torná-lo simples, agradável e muito dinâmico em nossas aplicações.

Assim reduzimos o nosso tempo com a manutenção dessas aplicações e ainda integramos as técnicas de programação orientada a objetos. Espero que tenham gostado das técnicas de *Globalization* e que passem a utilizá-las em seu dia-a-dia. Só para lembrar, esse recurso também está presente para aplicações *Dekstop* voltadas ao *.NET Framework*. Abraços e até o próximo artigo. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Listagem 9. Alterando o evento Page_Load do login.aspx

```
procedure TDefault.Page_Load(sender: System.Object;
  e: System.EventArgs);
var
  FTranslate:TTranslate;
  I: Integer;
begin
  FTranslate := TTranslate.Create;
  FTranslate.InicializeTranslate(Self.ClassName);
  for i := 0 to Controls.Count-1 do
    if Controls[i] is HtmlForm then
    begin
      FTranslate.TranslateControl(Controls[i]);
      break;
    end;
end;
```

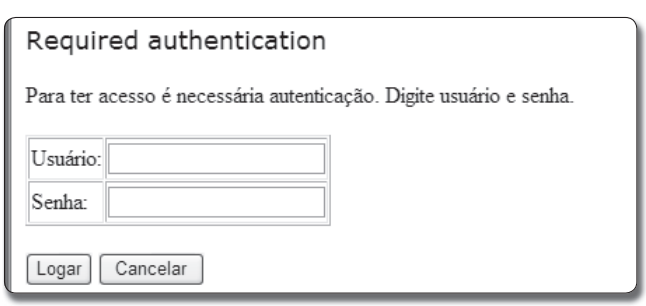


Figura 3. Primeiro Label traduzido pelo strings.en.resx

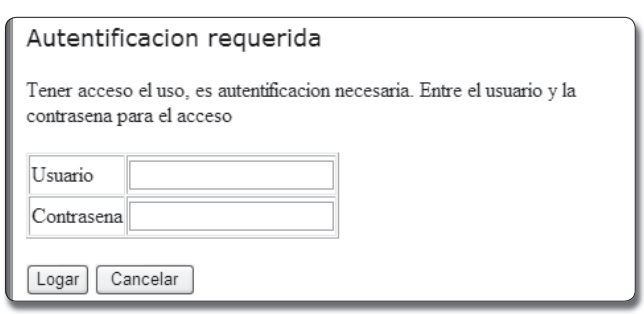


Figura 4. Aplicação sendo executada Espanhol

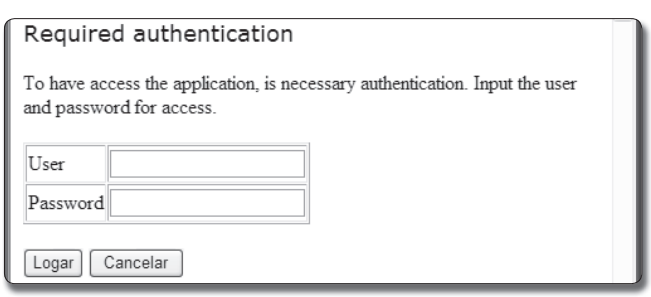


Figura 5. Aplicação sendo executada Inglês

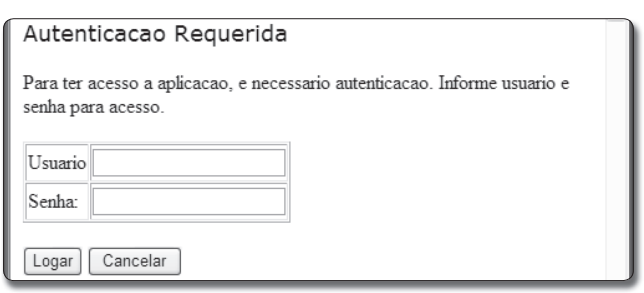


Figura 6. Aplicação sendo executada em Português

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Saiba como documentar seu código-fonte

Documentando propriedades e métodos com o RAD Studio 2007

Resumo DevMan

Veremos neste artigo como documentar nosso código-fonte de forma que seja possível vermos todas as informações e exemplos necessários para determinadas funções, assim fica mais fácil compreendermos nossos sistemas.

Nesse artigo veremos

- Como documentar corretamente o código fonte;
- Criando documentação.

Qual a finalidade

- A documentação é essencial para que outros programadores possam dar manutenção no seu código.

Quais situações utilizam esses recursos?

- Em todos os projetos se faz necessário uma boa documentação para que possamos melhorar ainda mais a codificação do aplicativo.

Rodrigo Otto Mostaert
otto3d@gmail.com
é Analista/Desenvolver Delphi desde sua versão 7. Possui diversos cursos pela TWM Tecnologia (atual parceria da Borland em Recife), pela Facilit e sólidos conhecimentos nas ferramentas CodeGear. Atualmente trabalha com Delphi desenvolvendo sistema orientado a objeto utlizando diversas camadas e banco de dados Oracle.

Neste artigo veremos como documentar nosso código fonte utilizando o RAD Studio 2007. É sabido que a cada nova versão, nosso Delphi vem com várias novidades e uma delas é o *Tooltip Help Insight*, presente desde o Delphi 8. O recurso permite, em tempo de desenvolvimento, obter informações sobre um determinado método ou propriedade no ato da codificação. Apenas na versão 2007 é que nós desenvolvedores podemos documentar cada propriedade ou método de nosso projeto. Na prática, veremos como documentar e fazer uso desse recurso para nos auxiliar no dia-a-dia.

Configurando o IDE

Primeiro teremos que habilitar nosso IDE pra exibir o painel de ajuda em tempo de desenvolvimento, para isso, acesse o menu *Tools>Options>Editor Options>Code Insight*. No *ComboBox Source file type* selecione a opção *Pascal* e logo em seguida marque a opção *Tooltip Help Insight* caso não esteja e clique em *Ok* **(Figura 1).**

Criando um projeto de teste

Pronto, nosso IDE já está habilitado, agora vamos à parte prática. Inicie um novo projeto em *File>New>VCL Forms*

Application – Delphi for Win32, lembrando que nosso exemplo também é válido para *.NET*. Salve o projeto como " ProjetoDoc.dproj" e a Unit principal de *UnitPrincipal.pas*. O próximo passo será criar uma classe de negócio chamada *TPessoa* com apenas duas propriedades: *Código* e *Nome*. Crie uma nova *Unit* através do menu *File>New>Unit – Delphi for Win32* salve-a com o nome *UnitPessoa.pas*. Agora iremos montar nossa classe, como mostra a **Listagem 1**.

Crie uma sessão *Uses* logo abaixo da sessão *Interface* e nela declare a *Unit SysUtils*. Em seguida declare um tipo como segue:

```
Type
  TPessoa = class(TObject)
  private
  public
  published
  end;
```

Na área *published*, faremos a declaração de duas propriedades: *Nome* e *Codigo*. *Codigo* será do tipo *Integer* lendo (*read*) em *FCodigo* e gravando (*write*) em *SetCodigo*. Já a propriedade *Nome* será uma *String* lendo em *FNome* e gravando em *SetNome*. Veja como ficam as declarações:

```
published
  property Codigo: integer read FCodigo
    write SetCodigo;
  property Nome: String read FNome
    write SetNome;
end;
```

Pressione *Ctrl + Shift + C* para que o Delphi crie para nós todos os cabeçalhos das *procedures* e também as variáveis necessárias para a implementação das propriedades. Seu código ficará semelhante ao mostrado no código da **Listagem 1**.

Para completar a implementação criaremos uma *function* chamada *ValidarData* para validar data como o próprio nome diz, apenas para fins didáticos. Portanto, declare a função como segue e, novamente, pressione *Ctrl + Shift + C* para criar o cabeçalho da função na área *Implementarion*. Essa função deve ser criada na sessão *public* da classe e seu conteúdo deve se parecer com a **Listagem 2**. Veja que estamos fazendo algo bem simples. Apenas recebemos a data em formato *String* e tentamos convertê-la para Data usando o método *StrToDateTime* da *Unit SysUtils*. Envolvemos o teste em um bloco *try..except* para que

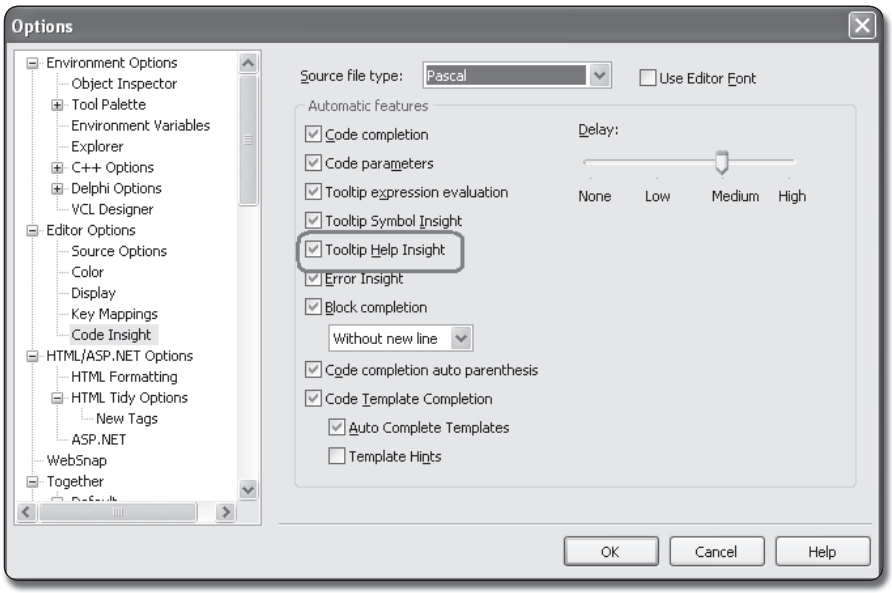


Figura 1. Habilitando o Tooltip Help Insight

Listagem 1. Criação da classe TPessoa

```
unit UnitPessoa;

interface

Type
  TPessoa = class(TObject)
  private
    FCodigo: integer;
    FNome: String;
    procedure SetCodigo(const Value: integer);
    procedure SetNome(const Value: String);
  public
  published
    property Codigo: integer read FCodigo write
      SetCodigo;
    property Nome: String read FNome write SetNome;
  end;


implementation

{ TPessoa }

procedure TPessoa.SetCodigo(const Value: integer);
begin
  FCodigo := Value;
end;

procedure TPessoa.SetNome(const Value: String);
begin
  FNome := Value;
end;

end.
```

Listagem 2. Criação da classe TPessoa

```
..

function TPessoa.ValidarData(Data: String): boolean;
begin
  Result := False;
  try
    StrToDateTime(Data);
    Result := True;
  except
    on EConvertError do
      EConvertError.Create('Data invalida!');
  end;
end;

end.
```

possamos retornar uma mensagem ao usuário caso ocorra uma exceção.

```
  ..
  public
    function ValidarData(Data: String):
boolean;
  published
  ..
```

Incluindo as TAG's de documentação no código

Para documentar nossas classes no *RAD Studio 2007* usamos o conceito de *Tag's*, semelhante ao HTML/XML. Existem várias *Tag's* para documentar nossa estrutura, tais como: *descrição, exemplo de código, parâmetro* ("nome e tipo"), tipo do retorno de um método e sua descrição, que tipo de exceção poderá ser levantada, uso de outros métodos, entre outras. Cada *Tag* é aplicada em diferentes situações. Para iniciá-las basta colocar três barras, a *tag* desejada, valor ("descrição") e finalizá-la, como mostrado a seguir:

```
/// <tag>Valor</tag>
```

Neste artigo iremos aprender as principais e mais utilizadas *Tag's* de documentação. As mais comuns são:

- Descrição = /// <summary>Descrição do método ou propriedade </summary>;
- Exemplo de código = /// <example>Descricao <code>Código exemplo</code></example>;
- Parâmetro= /// <param name="Nome"> Descrição do parâmetro</param>;
- Retorno = /// <returns>Descrição do retorno do método</returns>;
- Exceção = /// <exception>Descrição do método ou propriedade</exception>;
- Observação = /// <remark>Descrição do método ou propriedade</remark>;
- Parágrafo = /// <para>Parágrafo x</para>;
- Destacar palavra ou frase = /// <c> Palavra ou frase </c>;

> **Nota**: As *Tag's* no *Delphi* funcionam como se fosse *tag's* de formatação no *XML*: todas que abrirmos devem ser fechadas.

Vejamos na prática como isso funciona. Após criarmos a nossa classe, vamos agora implementar a documentação da propriedade *Codigo*. Localize a declaração desta *property* e digite o código da **Listagem 3**.

Observe que digitamos uma série de comentários antes da declaração da *property Codigo*. Esses comentários são, na verdade, nossas *tag's* e conseqüentemente a documentação da propriedade em questão. Note também que há uma endentação como se fosse código Delphi ou ainda HTML. Iniciamos uma *tag* de descrição *<summary>* e em seguida *<example>*. Dentro de *<example>* colocamos a *tag <code>* que recebe o trecho de código fonte para exemplificar o uso da propriedade *Codigo*. Por fim declaramos a *<returns>* que é a *tag* responsável por mostrar os retornos da função. Em nosso caso não haveria necessidade, já que estamos falando de propriedade e não de funções ou procedimentos.

Finalizada a digitação da documentação, retorne ao formulário principal e simule a codificação de uma variável do tipo *TPessoa*, como segue:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Pessoa: TPessoa;
begin
  Pessoa.cod
end;
```

Ao pressionar *Ctrl + Barra* para ativar o *Code Insight* do Delphi notamos que uma pequena caixa aparece ao lado do editor de códigos. Essa caixa nada mais

Listagem 3. Documentação da propriedade Código

```
..
  published
    /// <summary>
    ///   <example>Exemplo de atribuição de
      valor a propriedade Código
    ///     <code>
    ///       varPessoa.Codigo := Valor;
    ///     </code>
    ///   </example>
    /// </summary>
    /// <returns>
    ///   Retorna um valor inteiro com o código
      da <c>Pessoa</c>
    /// </returns>
    property Codigo: integer read FCodigo write SetCodigo;
..
```

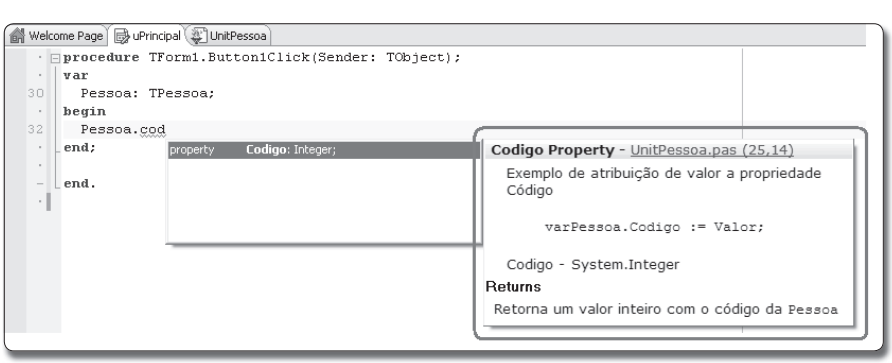
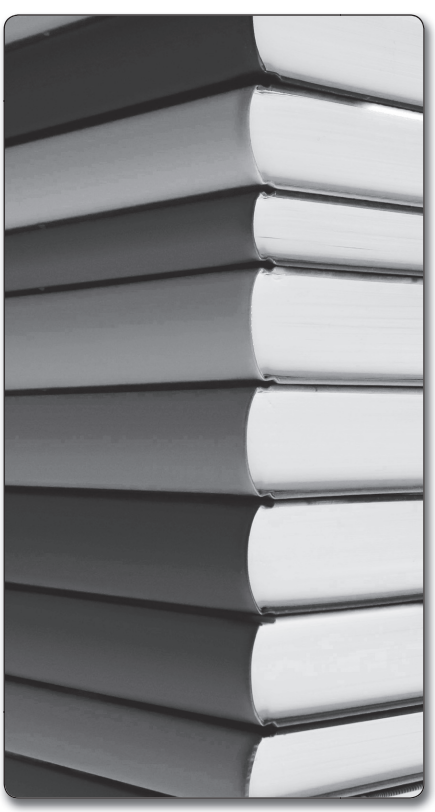


Figura 2. Documentação da propriedade Código sendo exibida



é do que a documentação que criamos anteriormente. Veja na **Figura 2**.

Muito bem, agora vamos criar a documentação de um método, definir seus parâmetros, as exceções que podem ser levantadas, funções externas que o método usa, retorno etc. Volte a *UnitPessoa.pas* e codifique o método *ValidarData* de acordo com a **Listagem 4**.

Dessa vez estamos incluindo outras *tag's* além das que já vimos anteriormente. Aqui vemos o uso de *<param name>* usado para mostrar um determinado parâmetro. E *<exception>* que mostra quais possíveis exceções possam ocorrer. Retorne a *Unit* principal e simule o uso da função *ValidarData* e veja o quadro que aparece logo ao lado do *Code Insight* na **Figura 3**.

Podemos usar quantas *tag's* forem necessárias para documentar nosso código. Basta usar a sintaxe correta.

Nota do DevMan

Documentar os tipos de exceção que um método pode levantar é uma excelente prática, pois permite que o método chamador possa saber como incluir os tratamentos corretos de acordo com o tipo em uma sessão try except.

Conclusão

O exemplo mostrado tem como objetivo mostrar uma das novidades que o *RAD Studio 2007* traz pra nós desenvolvedores. Hoje para um sistema pequeno, médio ou de grande porte, é necessário uma boa e clara documentação. Além dos editores de texto, onde documentamos os requisitos, casos de uso, e outros. Podemos também documentar cada método ou propriedade, assim, agilizando o uso de cada uma delas. Até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Listagem 4. Documentação do metodo ValidarData

```
  public
    /// <summary>
    ///   <example> Validacao de data
    ///     <code>
    ///       varValidado := varPessoa.ValidarData('01/01/2008');
    ///     </code>
    ///   </example>
    /// </summary>
    /// <param name="Data: String">Informe a data
      para validação</param>
    /// <returns>Retorna verdadeiro ou falso</returns>
    /// <exception cref="EConvertError">O valor
      informado não é uma data valida</exception>
    /// <remarks>
    ///   A validacao usa <see cref="StrToDateTime" />
      durante a operacao de validacao
    ///   com o parametro <paramref name="Data"/>.
    /// </remarks>
    function ValidarData(Data: String): boolean;
..
```

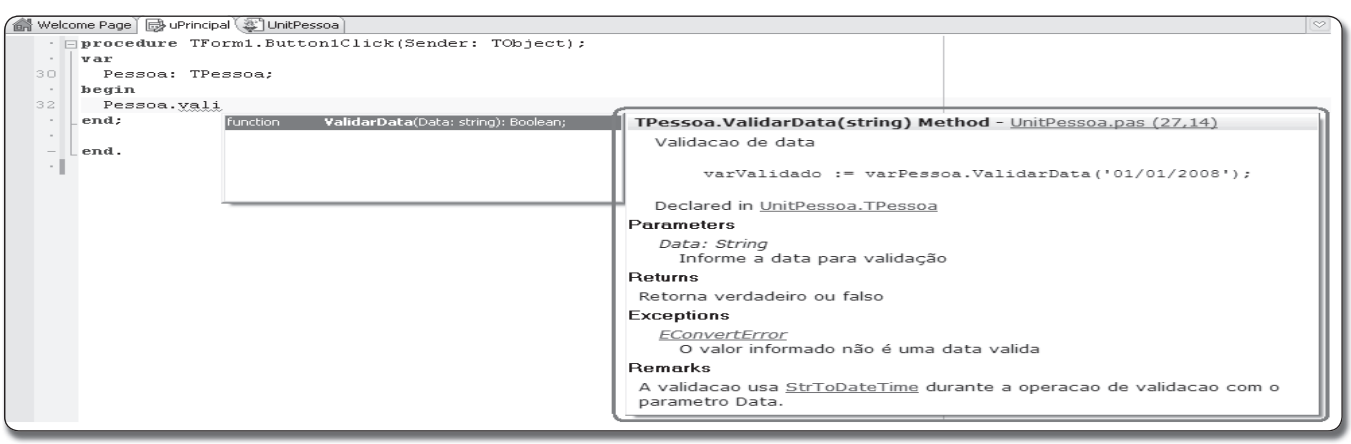
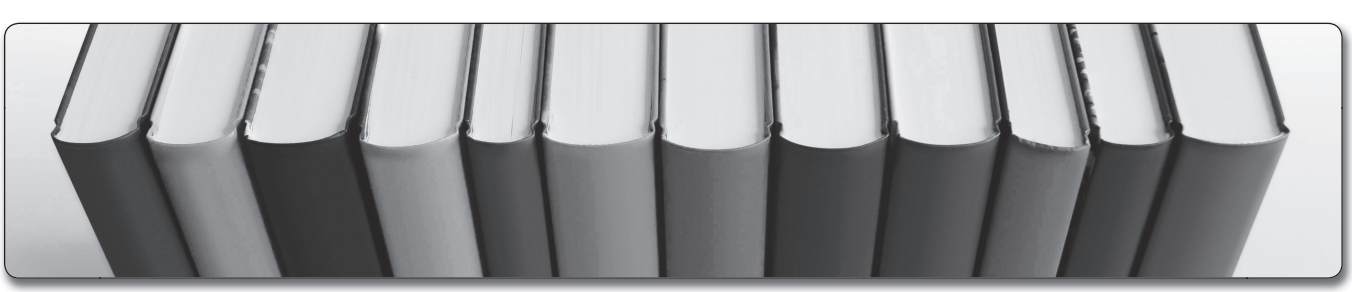
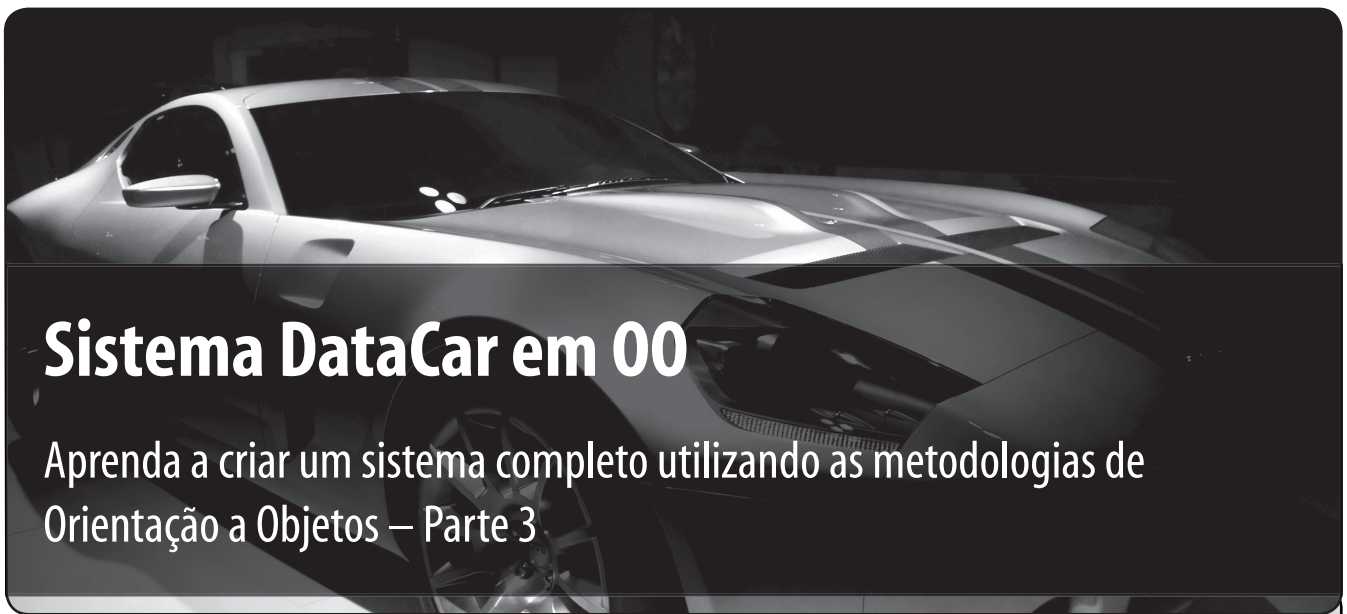


Figura 3. Documentação para o método ValidarData



Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET



Sistema DataCar em OO

Aprenda a criar um sistema completo utilizando as metodologias de Orientação a Objetos – Parte 3

Paulo Roberto Quicoli
pauloquicoli@gmail.com
é analista e programador da Control-M Informática. Trabalha com Delphi, desde sua primeira versão, e Firebird desenvolvendo aplicações cliente-servidor. Formado em Tecnologia de Processamento de Dados pela FATEC, na cidade de Taquaritinga/SP. Blog: http://pauloquicoli.spaces.live.com

Resumo DevMan

Veremos como aplicar todos os conceitos de Orientação a Objetos ensinada ao longo dessa série de artigos. A aplicabilidade será produzida em forma de cadastros simples e intermediários.

Nesse artigo veremos

- Criação dos principais cadastros do sistema;
- Aplicabilidade na prática dos conceitos de OO;
- Criação de cadastros utilizando a metodologia MVC.

Qual a finalidade

- Veremos como aplicar na prática todos os conceitos da Orientação a Objetos para montagem de cadastros básicos;

Quais situações utilizam esses recursos?

- Aplicável a todo sistema de gerenciamento de venda de carros.

Após termos construído nosso modelo de classes que representa nosso negócio e termos implementado uma forma de persistir nossas informações e ter testado tudo isso, chegou a hora de desenvolvermos a interação do usuário com o núcleo do sistema.

A idéia aqui é implementar os formulários necessários utilizando as boas práticas da orientação a objetos, contudo sem perder o suporte *RAD* que o *Delphi* nos oferece. Isso quer dizer que vamos mesclar, ou até porque não dizer, adaptar o padrão *MVC* ao *RAD* do *Delphi*.

Menu principal

Vamos começar pelo menu principal. Abra o projeto DataCar 2008 no *Delphi* e no projeto adicione um novo formulário, para isso clique com o botão direito do mouse sobre o nome do projeto. No menu que surgir escolha *Add New>Form*. Salve o projeto e dê o nome de *PrincU.dpr* para a *unit* e *PrincF.pas* para o formulário.

Adicione agora um *TMainMenu* e um *TActionList*. No *TActionList* adicione uma nova *action*, dando-lhe o nome de *acCadCidade*, altere seu *caption* para *Cidades* e na propriedade *category* digite *Cadastros*.

Agora no *TMainMenu* adicione um novo item chamado *Cadastros*. Nesse item *Cadastros* vamos relacionar os cadastros básicos do sistema, como *cidades*, *cores*, *acessórios*, *marcas* etc. Por enquanto vamos criar o subitem *Cidades* e ligaremos nele a *action acCadCidade*. Adicione também um *TToolBar* e um *TImagelist*. Nesse *TToolBar* posteriormente vamos adicionar um atalho para as rotinas mais corriqueiras do sistema. O seu formulário principal provavelmente deve estar parecendo com o da **Figura 1**.

Criando a rotina de cadastro de cidades

Vamos iniciar criando um formulário base de cadastro. Isso agiliza e padroniza nosso desenvolvimento. Observe que para isso vamos utilizar herança. Crie um novo formulário e salve-o como *cadBaseU.pas* e mude a propriedade *name* para *cadBaseF*.

Desabilite os botões de maximizar e minimizar e mude a borda para *single*. Adicione um *TToolBar* e um *TImageList*. Altere as propriedades *Height* e *Width* do *TImage* para 24. Ligue a propriedade *Images* do *TToolBar* ao *TImageList*. Clique com botão direito do mouse sobre o *TToolBar* e adicione 4 *ToolButtons*, ajustando sua propriedade *Name* para *tbAdicionar*, *tbEditar*, *tbExcluir*, *tbFechar* respectivamente. Ajuste a propriedade *caption* de cada um deles de acordo com sua função e adicione ao *ImageList* imagens que representam essas operações. Depois disso adicione um *TDBGrid* ao formulário. Na **Figura 2** podemos ver como pode ficar nosso formulário de cadastro base.

Como já mencionado no início deste artigo vamos utilizar o padrão *MVC* para ligar nossos formulários ao nosso modelo de classes. Não vou detalhar aqui toda a teoria do padrão *MVC* visto que isso já foi muito bem feito em edições anteriores por outros autores e até por mim mesmo. Contudo quero apenas lembrar que o padrão *MVC* introduz três partes distintas:

- *Model* – É a representação da informação que a aplicação utiliza;
- *View* – É a representação gráfica do *Model*. Por exemplo, um formulário expondo as informações da cidade. Uma *View* está sempre ligada a um *model* e sabe como exibi-lo ao usuário e para um determinado *Model* podem existir várias *views*. Além disso, também pode ser composta por várias *Views*;
- *Controller* – É responsável em mapear as ações do usuário em ações do sistema, por exemplo, se um botão ou item de menu é clicado é ele que deve definir como a aplicação responderá.

Na definição conceitual também se define a utilização de um *DTO* (*Data Transfer Object*), que é responsável por transferir os dados da *view* para o *model* ou vice-versa. Hoje vamos fazer uso desse *DTO* de tal forma que a *View* não tenha ligação direta com nossas classes que representam o negócio.

Implementando o DTO

Nossos objetos *DTO* serão nada mais do que *ClientDatasets* que representam os objetos de negócio, nossas entidades. Dessa forma não perdemos os recursos valiosos que o *Delphi* oferece para construção de interfaces gráficas.

Vamos adicionar um novo *DataModule* ao projeto e chamá-lo de *dmCadastros*. Nele vamos adicionar um *TClientDataset*, então alteramos seu nome para *cdsCidade* e nele criamos campos que representam a estrutura da nossa classe cidade. Na **Figura 3** vemos como ficou nosso *DataModule*, já com um *TDataSource* ligado ao *cdsCidade*.

Figura 2. Cadastro base

Figura 1. Menu principal do sistema

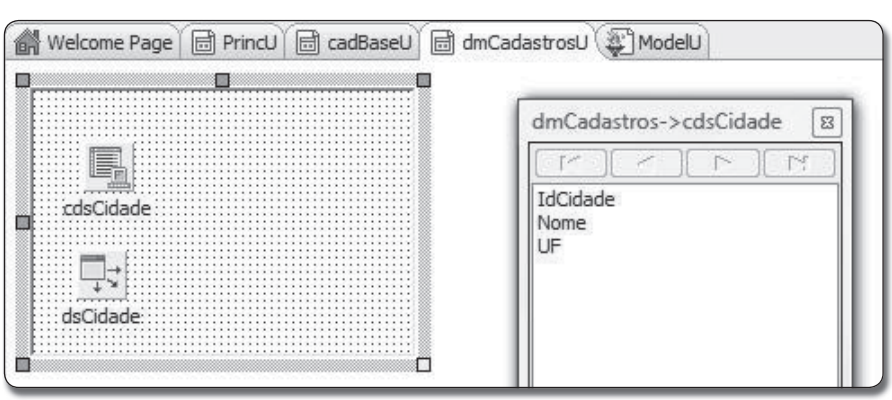


Figura 3. dmCadastros

Listagem 1. Interfaces do MVC

```
unit MVCInterfaces;
interface
type
  IView = interface
    ['{A1F411E9-294D-444D-9D5B-1D6AC9988819}']
    procedure Initialize;
  end;
  IController = interface
    ['{E4BD8853-F6C8-4BD4-B19D-D70B156BD712}']
    procedure Initialize;
  end;
implementation
end.
```

Listagem 2. Classe Controller

```
TCadCidadeController = class(TInterfacedObject, IController)
private
  daoCidade: ICidadeDao;
public
  constructor Create; reintroduce;
  procedure Initialize;
  procedure Insert;
  procedure Delete;
  procedure Edit;
  procedure Save;
  procedure Finalize;
  procedure Cancel;
end;
```

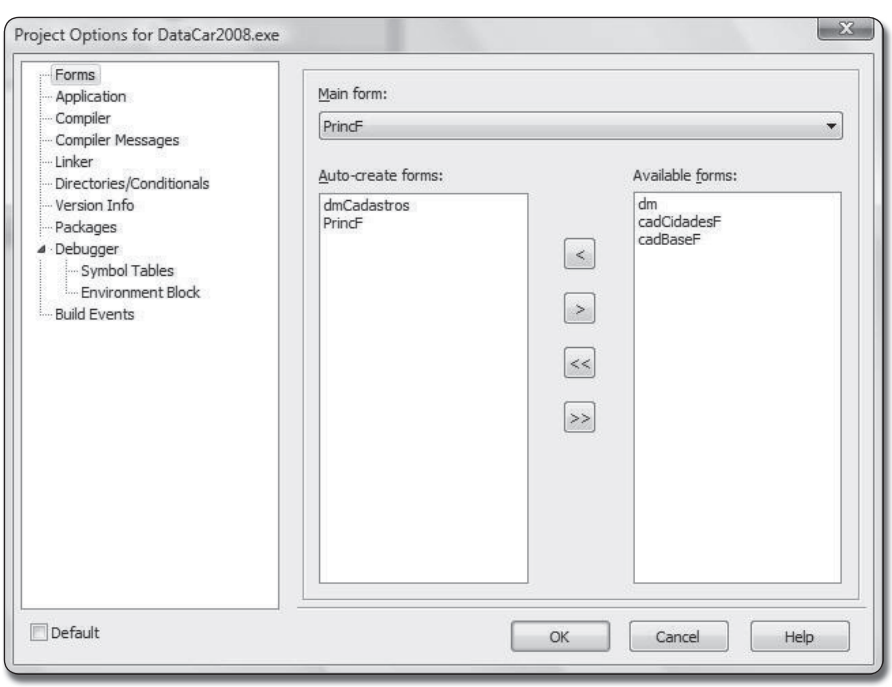


Figura 4. Opções do projeto

Aplicando o MVC

Como já foi mencionado, o *MVC* introduz três partes, o *Model* a *View* e o *Controller* e como também já foi mencionado, vamos fazer uma adaptação do *MVC* às nossas necessidades e por isso vamos implementar duas interfaces que irão representar o *Controller* e a *View*. Adicione uma nova *unit* ao projeto, chamando-a de *MVCInterfaces.pas* e nela adicione o código da **Listagem 1**.

Com isso acabamos de criar duas *interfaces*. Elas serão responsáveis pela *View* e o *Controller*. Adicione uma nova *unit* ao projeto, salvando-a como *cadCidadeControllerU.pas*. Nela adicione no uses as *units MVCInterfaces*, *DaoU*, *DaoDbx*, *ModelU* e *dmCadastrosU*. Vamos criar o *controller* para o cadastro de cidades. Crie uma nova classe conforme **Listagem 2.**

Veja que nosso *controller* expõe alguns métodos de *CRUD* (*create*, *retrieve*, *update*, *delete*). Nesses métodos nosso *DTO*, representado pelo *cdsCidade* do *datamodule dmCadastros*, é manipulado, conforme vemos na **Listagem 3**.

Quero chamar a atenção aqui para alguns métodos. No *Create* do *controller* estamos instanciando um objeto que implementa *ICidadeDao*. Nesse momento poderíamos buscar em alguma configuração do sistema qual *DAO* que deveria ser instanciado, se, por exemplo, um *DAO* que utiliza *DBX* ou um que utiliza *IBX*. Como neste exemplo criamos apenas um *DAO* que utiliza *DBX* já que o instanciamos diretamente.

Veja também o método *Initialize*. Nele, carregamos o conteúdo do banco de dados, através do *DAO*, para o nosso *DTO* que é um *ClientDataSet*. Posteriormente esse *ClientDataSet* será utilizado por uma *View*. E finalmente observe o método *Save*. Nele passamos para um objeto *TCidade* o valor correspondente do registro do *ClientDataSet*.

Criando a View

Com o *controller* pronto vamos criar nossa *view*. Adicione ao projeto um novo formulário que herde de *TcadBaseF*. Salve o formulário como *cadCidadesViewU.pas*. Configure as propriedades *Name* e *Caption* respectivamente para *cadCidadesF* e "Cadastro de Cidades". Acesse o menu *Project>Options* e o deixe como na **Figura 4**.

Listagem 3. Implementação do controller

```
procedure TCadCidadeController.Cancel;
begin
  dmCadastros.cdsCidade.Cancel;
end;
constructor TCadCidadeController.Create;
begin
  inherited Create;
  daoCidade := TCidadeDao.Create;
  dmCadastros.cdsCidade.CreateDataSet;
end;
procedure TCadCidadeController.Delete;
var
 objCidade: TCidade;
begin
  try
    try
      objCidade := TCidade.Create;
      objCidade.Id := dmCadastros.cdsCidadeIdCidade.AsInteger;
      daoCidade.Excluir(objCidade);
      dmCadastros.cdsCidade.Delete;
    except
      on e:exception do
      begin
         dmCadastros.cdsCidade.Cancel;
         raise Exception.Create(e.Message);
      end;
    end;
  finally
    FreeAndNil(objCidade);
  end;
end;
procedure TCadCidadeController.Edit;
begin
  dmCadastros.cdsCidade.Edit;
end;
procedure TCadCidadeController.Finalize;
begin
  dmCadastros.cdsCidade.Close;
end;
procedure TCadCidadeController.Initialize;
var
  cidades: TObjectList;
  I: Integer;
begin
  cidades := daoCidade.ListarTodasCidades;
  try
    dmCadastros.cdsCidade.CreateDataSet;
  except
    dmCadastros.cdsCidade.Open;
    dmCadastros.cdsCidade.EmptyDataSet;
  end;
  IF cidades <> nil then
  for I := 0 to cidades.Count - 1 do
  begin
    dmCadastros.cdsCidade.Append;
    dmCadastros.cdsCidadeIdCidade.AsInteger := TCidade(cidades[I]).Id;
    dmCadastros.cdsCidadeNome.AsString := TCidade(cidades[I]).Nome;
    dmCadastros.cdsCidadeUF.AsString := TCidade(cidades[I]).UF;
    dmCadastros.cdsCidade.Post;
  end;
end;
procedure TCadCidadeController.Insert;
begin
  dmCadastros.cdsCidade.Append;
  dmCadastros.cdsCidadeIdCidade.AsInteger := 0;
end;
procedure TCadCidadeController.Save;
var
  objCidade: TCidade;
begin
  objCidade := TCidade.Create;
  objCidade.Id := dmCadastros.cdsCidadeIdCidade.AsInteger;
  objCidade.Nome := dmCadastros.cdsCidadeNome.AsString;
  objCidade.UF := dmCadastros.cdsCidadeUF.AsString;
  try
    try
      daoCidade.Salvar(objCidade);
      dmCadastros.cdsCidadeIdCidade.AsInteger := objCidade.Id;
      dmCadastros.cdsCidade.Post;
    except
    on e: exception do
      begin
        dmCadastros.cdsCidade.Cancel;
        raise Exception.Create(e.Message);
      end;
    end;
  finally
    FreeAndNil(objCidade);
  end;
end;
```

Listagem 4. View de cadastro de cidades

```
  TcadCidadesF = class(TcadBaseF, IView)
  private
    { Private declarations }
    FController: TcadCidadeController;
  public
    { Public declarations }
    constructor Create(Owner: TComponent; Controller:
      TcadCidadeController); reintroduce;
    procedure Initialize;
  end;

var
  cadCidadesF: TcadCidadesF;

implementation

{$R *.dfm}

{ TcadCidadesF }


constructor TcadCidadesF.Create(Owner: TComponent; Controller: TcadCidadeController);
begin
  inherited Create(Owner);
  FController := Controller;
end;

procedure TcadCidadesF.FormShow(Sender: TObject);
begin
  inherited;
  Initialize;
end;

procedure TcadCidadesF.Initialize;
begin
  FController.Initialize;
end;

procedure TcadCidadesF.tbFecharClick(Sender: TObject);
begin
  inherited;
  Close;
end;

procedure TcadCidadesF.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  inherited;
  FController.Finalize;
end;
```

Listagem 5. Chamando a View

```
procedure TPrincF.acCadCidadesExecute(Sender: TObject);
var
  Controller: IController;
begin
  Controller := TCadCidadeController.Create;
  try
    cadCidadesF := TcadCidadesF.Create(nil,Controller);
    cadCidadesF.ShowModal;
  finally
    FreeAndNil(cadCidadesF);
  end;
end;
```

Listagem 6. Excluindo uma cidade

```
procedure TcadCidadesF.tbExcluirClick(Sender: TObject);
begin
  inherited;
  if Application.MessageBox('Confirma exclusão da cidade selecionada?',
    'Atenção',MB_YESNO +MB_ICONQUESTION +
    MB_DEFBUTTON2) = ID_YES then
  begin
    try
      FController.Delete;
    except
    on e: Exception do
      begin
        ShowMessage(e.Message);
      end;
    end;
  end;
end;
```

Configurando conforme mostrado na figura anterior apenas o *dmCadastros* e o formulário principal serão criados automaticamente. Vamos adicionar agora ao *uses* da nossa *View* uma referência para *MVCInterfaces*, *dmCadastrosU* e *cadCidadeControllerU*. Altere a classe do formulário como na **Listagem 4**.

Na declaração da classe dizemos que nosso formulário implementa *IView* e portanto deve implementar o método *Initialize*. Depois, seguindo o padrão *MVC* adicionamos uma referência a um *controller*, referência essa que deve ser passada no construtor do formulário. Esse *controller* é utilizado por enquanto no método *Initialize* da própria *View* que por sua vez é chamado no evento *onShow* do formulário. Não podemos esquecer de ligar o *dbGrid* ao *DataSource dsCidade*.

Para chamar nossa *View* vamos até o formulário principal e vamos implementar então a *Action* anteriormente criada, a *acCadCidades*. Na **Listagem 5** temos o método *onExecute* da *Action*.

Veja que criamos nosso *controller TCadCidadeController* e o passamos para o construtor da *view*, dessa forma mantemos um relacionamento entre *View* e *Controller*. Na **Figura 5** vemos nossa *View* em execução exibindo algumas cidades previamente cadastradas para teste.

Para terminar nosso cadastro temos que implementar as operações de *Adicionar*, *Editar* e *Excluir*.

Excluindo uma cidade

Ao clicar no botão *Excluir* chamamos o método *Delete* do *controller*, ele sabe como a exclusão deve ser feita. Veja na **Listagem 6**.

Inserindo/Editando uma cidade

Ao clicar no botão *Adicionar* ou *Editar* o *controller* será acionado para realizar

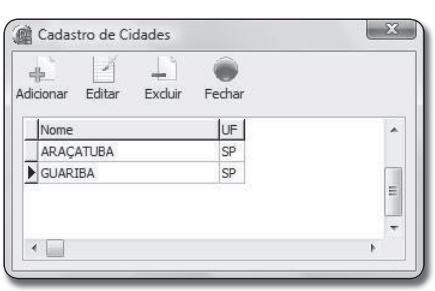


Figura 5. View cadastro de cidades

a tarefa específica e uma nova *View* será chamada, a *View* de edição. Semelhantemente como fizemos com a *View* de cadastro também vamos criar um formulário base para esse tipo de operação. Crie um novo formulário, salve sua *unit* como *EditBaseU.pas* e o deixe como na **Figura 6**.

Agora criamos um outro formulário herdando de *TEditBase*, este se chamará *EditCidadeF* e sua *unit EditCidadeU*. Na seção *uses* desse formulário adicionamos uma referência para *MVCInterfaces*, *dmCadastrosU* e *cadCidadeControllerU* e implementamos também um construtor que recebe um *Controller* como parâmetro, sem esquecer de indicar na declaração da classe que será implementada a interface *IView*. Veja **Listagem 7**.

E novamente chamamos o *controller* para realizar as operações de *Salvar* e *Cancelar*, conforme **Listagem 8**.

Agora podemos adicionar os *dbEdits* que vão permitir ao usuário. Neste momento fica faltando apenas chamar a *view* de edição a partir da *view* de cadastro de cidades. Adicionamos um referência à *unit EditCidadeU* e então instanciamos o formulário passando o *controller* no construtor. Na **Listagem 9** vemos como isso é feito.

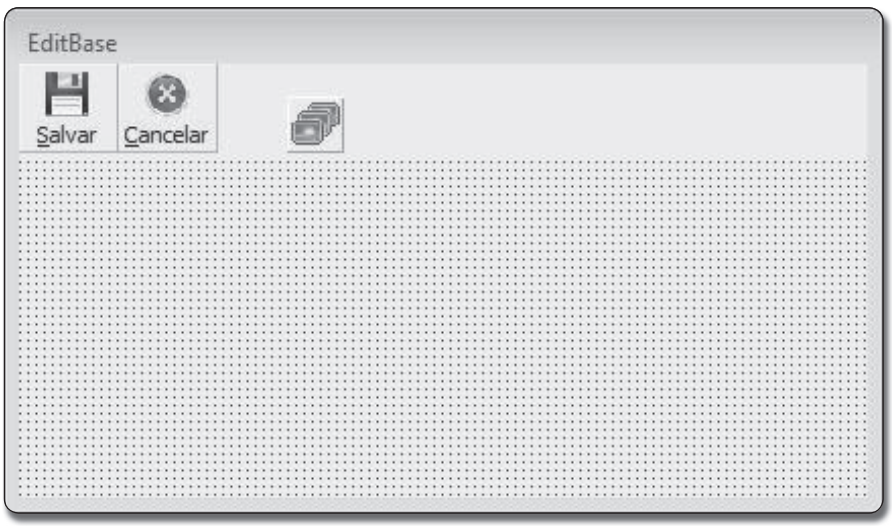


Figura 6. Formulário de edição

Listagem 7. View de Edição

```
type
TEditCidadeF = class(TEditBaseF, IView)
  private
    { Private declarations }
    FController: TcadCidadeController;
  public
    { Public declarations }
    constructor Create(Owner: TComponent; Controller: TcadCidadeController);
reintroduce;
    procedure Initialize;
  end;
var
  EditCidadeF: TEditCidadeF;
implementation
uses dmCadastrosU, cadCidadeControllerU;
{$R *.dfm}
{ TEditCidadeF }
constructor TEditCidadeF.Create(Owner: TComponent; Controller: TcadCidadeController);
begin
  inherited Create(Owner);
  FController := Controller;
end;
procedure TEditCidadeF.Initialize;
begin
  exit;
end;
```

Listagem 8. Salvando ou cancelando as informações

```
procedure TEditCidadeF.tbCancelarClick(Sender: TObject);
begin
  inherited;
  FController.Cancel;
  ModalResult := mrCancel;
end;
procedure TEditCidadeF.tbSalvarClick(Sender: TObject);
begin
  inherited;
  if Application.MessageBox('Confirma os dados da cidade?',
       'Atenção', MB_YESNO + MB_ICONQUESTION + MB_DEFBUTTON2) = ID_YES then
  begin
    try
      try
       SendMessage(dbeditUf.Handle,CM_EXIT,0,0);
       FController.Save;
      except
      on e:Exception do
        begin
          FController.Cancel;
          ShowMessage(e.Message);
        end;
      end;
    finally
      ModalResult := mrOK;
    end;
  end;
end;
```

Nota do DevMan

Quando um *TDBEdit* está ligado a um *TDataSource* e você altera o conteúdo desse *TDBedit*, o *TDBEdit* só envia para o *TDataSource* os dados que você alterou caso o *TDBEdit* perca o foco, ou seja, aquilo que você digita só vai para o *TDataSource* se você sair do campo através de um TAB ou passe o foco para outro controle. No nosso caso o *dbEditUF* não perde o foco porque ao terminarmos de digitar um valor, clicamos em um *TToolButton* que não recebe o foco e com isso o *dbEditUF* não envia os dados para o *TDataSource* ligado nele implicando que o valor digitado não é salvo. Para resolver essa situação enviamos uma mensagem ao *TDBEdit* que o notifica para que ele perca o foco e com isso as alterações sejam enviadas para o *TDataSource*. Veja como isso é feito antes do método *FController.Save* da Listagem 8.

E para alterar uma cidade apenas mudamos o método do *controller* de *Insert* para *Edit*, como na **Listagem 10**. Perceba que o código é bem simples. Apenas chamamos o método *Edit* do *Controller* e em seguida da instância.

Com isso terminamos o cadastro de cidades. Para os outros cadastros simples o procedimento é o mesmo. O que eu gostaria de destacar é que o formulário não sabe como os dados são salvos. Observe que ele não possui referência alguma aos objetos de negócio. Isso torna nosso formulário independente do restante do sistema. Na **Figura 7** temos a representação da interação do formulário com as outras classes.

Um cadastro mais complexo

O cadastro de modelos é um pouco mais complexo de se implementar visto que nele temos uma relação entre marca e modelo. Vamos começar adicionando um *ClientDataSet* ao *Data Module dmCadastros* e nele vamos criar campos que armazenem as informações sobre o modelo e sobre sua marca. Veja **Figura 8**.

Na **Listagem 11** temos o método *ListarTodosModelos* da classe *TModeloDao*. Nele podemos obter todos os modelos cadastrados. Observe que na geração do *SQL* realizamos um *join* entre *modelo* e *marca*.

Agora é a vez do *Controller*. Nosso *controller* de cadastro de modelos deve possuir alguns métodos a mais já que a partir desse cadastro é preciso listar as *marcas* e permitir que uma nova *marca* seja criada, caso não exista. Para desenvolver o *controller* adicionamos uma nova *unit* e a salvamos como *cadModeloControlleU.pas*, adicionamos referências a *MVCInterfaces*, *DaoU*, *DaoDbx* e *ModelU*. Como já foi citado vamos precisar listar todas as *marcas* também por isso vamos definir um método *InitializeMarcas*, sua implementação está na **Listagem 12**.

Veja que listamos todas as marcas usando nosso método *ListarTodasMarcas*, como o próprio nome sugere. Em seguida fechamos e criamos o *DataSet*. Um detalhe interessante, estamos testando o fechamento e criação do *DataSet* usando um bloco *try..except*, caso um erro ocorra significa que o *DataSet* já encontra-se criado, então o limpamos e o abrimos, e em seguida incluímos a nova *marca*. Os outros métodos deste *controller* seguem o padrão mostrado no início do artigo. Um pequeno detalhe existe no método *Save*. Como *TModelo* possui uma referência a

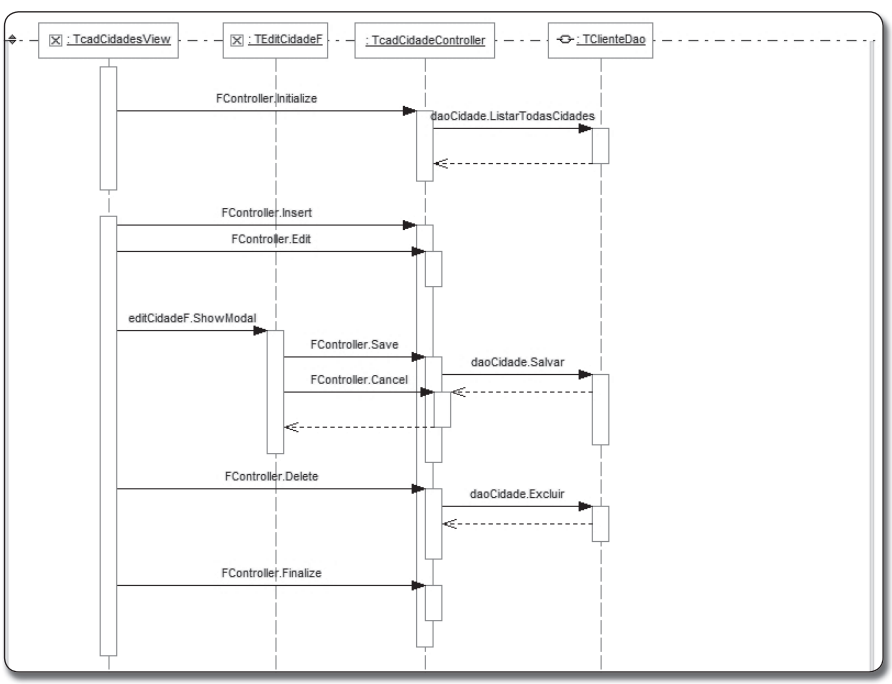


Figura 7. Diagrama de sequência

Listagem 9. Inserindo uma nova cidade

```
procedure TcadCidadesF.tbAdicionarClick(Sender: TObject);
begin
  inherited;
  FController.Insert;
  EditCidadeF := TEditCidadeF.Create(nil,FController);
  try
    EditCidadeF.ShowModal;
  finally
    FreeAndNil(EditCidadeF);
  end;
end;
```

Listagem 10. Editando uma cidade

```
procedure TcadCidadesF.tbEditarClick(Sender: TObject);
begin
  inherited;
  FController.Edit;
  EditCidadeF := TEditCidadeF.Create(nil, FController);
  try
    EditCidadeF.ShowModal;
  finally
    FreeAndNil(EditCidadeF);
  end;
end;
```

Figura 8. cdsModelo

TMarca é preciso que nós a forneçamos. Veja **Listagem 13**.

Para criar *View*, criamos como fizemos com outros cadastros, porém o método *Initialize* da *View* irá chamar os métodos *Initialize* e *InitializeMarcas* do *Controller* conforme a seguir:

```
FController.Initialize;
FController.InitializeMarcas;
```

A *View* de edição será um pouco diferente. Veja na **Figura 9**.

Observe que temos nessa *View* um *DBLookupComboBox*. Ele é configurado apenas para exibir as marcas por isso apenas as propriedades *ListSource*, *ListField* e *KeyField* são configuradas para acessar as informações de *cdsMarca*. No evento *onShow* desse formulário chamamos o método *Initialize* da *View*, que por sua vez executa o método *InitializeMarcas* do *Controller* (**Listagem 14**).

Vimos na **Listagem 14** que o formulário *EditModeloF* ao ser exibido preenche o *cdsMarca* e conseqüentemente o *DBLookupComboBox* e então posicionamos o *KeyValue* para o valor que foi relacionado ao *Modelo* que está sendo editado. Dessa forma a marca correta é exibida para o usuário. Para salvarmos esse modelo é preciso obter o *id* da *marca* selecionada e passá-la ao objeto modelo. Veja **Listagem 15**.

E se ao criar um novo *modelo*, a *marca* que ele possui ainda não está cadastrada? É uma boa prática permitir ao usuário a criação dessa *marca* ali mesmo, sem precisar ir a outro formulário. Essa é a função do botão *btCadMarca* posicionado logo após o *DBLookupComboBox*. Ao clicar nele chamamos a *View* de edição de marcas (**Listagem 16**).

Conclusão

Nesta parte do artigo fizemos os cadastros básicos do sistema utilizando o

Listagem 11. Método ListarTodosModelos

```
function TModeloDao.ListarTodosModelos: TObjectList;
var
 ListaObjetos: TObjectList;
 Modelo: TModelo;
begin
  with SqlQuery, SQL do
  begin
    Close;
    Clear;
    Add('SELECT MODELO.ID_MODELO, MODELO.NOME, ');
    Add('MARCA.ID_MARCA, MARCA.NOME AS MARCA, MARCA.LOGOTIPO FROM MODELO');
    Add('JOIN MARCA ON (MARCA.ID_MARCA = MODELO.ID_MARCA)');
    Add('ORDER BY MODELO.NOME');
    try
      try
        Open;
        if IsEmpty then
           ListaObjetos := nil
        else
        begin
          ListaObjetos := TObjectList.Create;
          while not SqlQuery.Eof do
          begin
            Modelo := TModelo.Create;
            Modelo.Id := SqlQuery.FieldByName('ID_MODELO').asInteger;
            Modelo.Nome := SqlQuery.FieldByName('NOME').asString;
            Modelo.Marca.Id := SqlQuery.FieldByName('ID_MARCA').asInteger;
            Modelo.Marca.Nome := SqlQuery.FieldByName('MARCA').AsString;
            Modelo.Marca.Logotipo := FieldByName('LOGOTIPO').AsString;
            ListaObjetos.Add(Modelo);
            Next;
          end;
        end;
      except
      on e:exception do
      begin
        raise Exception.Create('Falha ao recuperar modelos'+ #13+
          'Mesagem original: '+ e.Message);
        ListaObjetos := nil;
      end;
      end;
    finally
      SqlQuery.Close;
      result := ListaObjetos;
    end;
  end;
end;
```

Listagem 12. InitializeMarcas

```
procedure TCadModeloController.InitializeMarcas;
var
  Marcas: TObjectList;
  I: integer;
begin
  Marcas := daoMarca.ListarTodasMarcas;
  try
    dmCadastros.cdsMarca.Close;
    dmCadastros.cdsMarca.CreateDataSet;
  except
    dmCadastros.cdsMarca.EmptyDataSet;
    dmCadastros.cdsMarca.Open;
  end;
  if Marcas <> nil then
    for I := 0 to Marcas.Count - 1 do
    begin
      dmCadastros.cdsMarca.Append;
      dmCadastros.cdsMarcaIdMarca.AsInteger := TMarca(Marcas[I]).Id;
      dmCadastros.cdsMarcaNome.AsString := TMarca(Marcas[I]).Nome;
      dmCadastros.cdsMarca.Post;
    end;
end;
```

padrão *MVC* de uma forma adaptada à nossa realidade. Ao procurar utilizar um padrão devemos entender exatamente o que ele pode nos oferecer e sermos capazes o suficiente para decidir qual é a melhor maneira de implementá-lo e utilizá-lo. Foi exatamente isso que fizemos e com isso assimilamos idéias que serão utilizadas para os próximos passos do nosso sistema, o registro de veículos para venda, veículos procurados, venda e relatórios. Fique ligado. ●

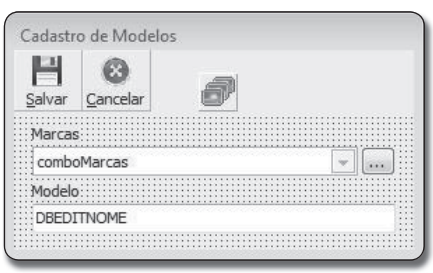


Figura 9. View de edição de modelos

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Listagem 13. Método save

```
procedure TCadModeloController.Save;
var
  objModelo: TModelo;
  objMarca: TMarca;
begin
  objModelo := TModelo.Create;
  objModelo.Id := dmCadastros.cdsModeloIdModelo.AsInteger;
  objModelo.Nome := dmCadastros.cdsModeloNome.AsString;
  objMarca := TMarca.Create;
  objMarca.Id := dmCadastros.cdsModeloIdMarca.AsInteger;
  objModelo.Marca := objMarca;
  try
    try
      daoModelo.Salvar(objModelo);
      dmCadastros.cdsModeloIdModelo.AsInteger := objModelo.Id;
      dmCadastros.cdsModelo.Post;
    except
    on e: exception do
    begin
      dmCadastros.cdsModelo.Cancel;
      raise Exception.Create(e.Message);
    end;
    end;
  finally
    FreeAndNil(objModelo);
  end;
end;
```

Listagem 14. Inicialização do formulário EditModeloF

```
procedure TEditModeloF.FormShow(Sender: TObject);
begin
  inherited;
  Initialize;
  comboMarcas.KeyValue := dmCadastros.cdsModeloIdMarca.AsInteger;
end;
procedure TEditModeloF.Initialize;
begin
  FController.InitializeMarcas;
end;
```

Listagem 15. Salvando um modelo

```
procedure TEditModeloF.tbSalvarClick(Sender: TObject);
begin
  inherited;
  if Application.MessageBox('Confirma os dados do modelo?',
    'Atenção', MB_YESNO + MB_ICONQUESTION + MB_DEFBUTTON2) = ID_YES then
  begin
    try
      try
        dmCadastros.cdsModeloIdMarca.AsInteger := comboMarcas.KeyValue;
        dmCadastros.cdsModeloMarca.AsString := dmCadastros.cdsMarcaNome.AsString;
        SendMessage(dbeditNome.Handle,CM_EXIT,0,0);
        FController.Save;
      except
      on e:Exception do
      begin
        FController.Cancel;
        ShowMessage(e.Message);
      end;
      end;
    finally
      ModalResult := mrOK;
    end;
  end;
end;
```

Listagem 16. Criando uma marca a partir do cadastro de modelos

```
procedure TEditModeloF.btCadMarcaClick(Sender: TObject);
var
 ControllerMarca: TCadMarcaController;
begin
  inherited;
  ControllerMarca := TCadMarcaController.Create;
  ControllerMarca.Edit;
  EditMarcaF := TEditMarcaF.Create(nil,ControllerMarca);
  try
    EditMarcaF.ShowModal;
  finally
    FreeAndNil(EditMarcaF);
    FController.InitializeMarcas;
  end;
end;
```

Já pensou em hospedar todos os seus sites em um único plano de hospedagem?

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe!

RVSTMEDIA

PLANO PROFISSIONAL 1

3 domínios independentes
50 GB de transferência
30 GB de e-mails com SSL
1 GB de espaço
Painel de controle
ASP, ASP.NET 3.5 E PHP 5
Access ilimitado
3 bancos MYSQL 5
1 banco SQL Server 2005 Express

Tudo isso por apenas R$ **25,90** por mês

Na Hospedix você ainda ganha registro de domínio (.com .net .org ou .info) grátis e isento de taxas anuais enquanto for cliente.

Acesse agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

hospedix.com.br

HOSPEDIX
HOSPEDAGEM PROFISSIONAL

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Introdução à criação de componentes

Aprenda os conceitos e como construir seu primeiro componente

Nesse artigo veremos

- Criação de componentes;
- Definindo métodos;
- Definindo propriedades.

Qual a finalidade

- Permitir a criação de componentes para aumento de produtividade e reaproveitamento de código.

Quais situações utilizam esses recursos?

- Códigos que se repetem em várias aplicações precisam ficar em um local centralizado, facilitando a manutenção e utilização. Componentes podem ser utilizados em qualquer tipo de aplicação.

A natureza *RAD* do *Delphi* deve muito a um conceito bastante conhecido pelos desenvolvedores: componentes. Você abre o *IDE* e tem à disposição componentes prontos para construir aplicações nas mais variadas áreas de desenvolvimento: Internet, Banco de Dados, *User Interface*, Web etc. Experimente navegar pela paleta de componentes e veja a variedade de recursos.

Neste artigo, veremos a anatomia de um componente, como são estruturados internamente, para que servem, e de praxe como criar nosso primeiro componente *Delphi* totalmente funcional.

Guinther Pauli
guinther@devmedia.com.br
é Bacharel em Sistemas de Informação, desenvolvedor Delphi há 13 anos, certificado pela Borland em Delphi: Web, 6, 7, 2005, 2006 e Kylix, e pela Microsoft: MCP, MCAD e MCSD. NET. É editor-chefe das revistas ClubeDelphi e .net Magazine.

O que é um componente

Componentes estão intimamente ligados à Programação Orientada a Objetos (*POO*). Na verdade, um componente é um objeto. Ou melhor, uma classe. Para quem está começando, pense numa classe como uma "receita de bolo", e os objetos os "bolinhos" que você pode criar com ela. No *Delphi*, temos inúmeras classes, agrupadas no que chamamos de *VCL* (*Visual Component Library*). A *VCL* é sem dúvida uma das responsáveis por todo o sucesso do *Delphi* até hoje.

Quem já programou em linguagem estruturada, como por exemplo, o antigo *Basic*, deve se lembrar das famosas rotinas. No Pascal também tínhamos esse conceito. Tudo o que precisasse

ser reaproveitado em vários prontos da aplicação, era movido para um procedimento, que podia então ser invocado inúmeras vezes. Surgia o reaproveitamento de código. O procedimento podia receber parâmetros, e assim se tornar mais flexível.

Com o surgimento da *POO*, surgiram as classes, que a grosso modo, se parecem com os procedimentos da linguagem estruturada, porém tendo inúmeras vantagens: encapsulamento, polimorfismo, herança etc. Então, classes são formas de reaproveitarmos código em nossas aplicações (em várias delas). E um componente, na verdade é uma classe que torna o trabalho ainda mais fácil, adicionando uma mistura *RAD* à receita.

Classes e componentes podem ser herdados de outros, isto é, eles podem aproveitar todo o código incluído em uma classe base e adicionar uma certa funcionalidade. Se você quiser montar um carro, por exemplo, provavelmente não vai construir todas as peças do zero, vai reaproveitar algumas, incrementar, reunir outras, e juntando tudo, temos o produto final. Assim funcionam as aplicações *OO* e com componentes.

No *Delphi*, criar um componente é muito simples. O primeiro passo é pôr em mente: o que eu preciso fazer, para identificar a classe base a ser herdada. Segundo: o que a mais esse meu componente irá fazer, aí você codifica. A má notícia é que a criação (não a utilização) de componentes no *Delphi* é feita de forma totalmente braçal, via código.

Antes de criar componentes, é preciso entender como funciona o coração da *VCL* do *Delphi*. Observe na **Figura 1** que é a mãe de todas as classes é *TObject*. Essa classe especifica como todos os objetos do *Delphi* devem se comportar, além de adicionar importantes funcionalidades de *RTTI* (*Runtime Type Information*). *TPersistent* é responsável por serializar objetos. Na prática, quando configurar um componente e fechar o *Delphi*, ao retornar, tudo o que configurou nele estará lá. *TComponent* serve de base para todos os componentes. *TControl* é uma classe abstrata (que não pode ser criada) que serve de base para todos os controles visuais, como botões, caixas de edição, painéis. A seguir temos duas importantes subdivisões. *TWinControl* são classes que apenas encapsulam controles já definidos pelo *Windows*, como *botões*. A *VCL* não faz o botão, apenas cria uma classe que envolve o botão do *Windows* já pronto (conhecidas como classes *wrapper*). Se você quiser construir seu próprio controle visual do zero, incluindo como pintá-lo, deverá herdar de *TGraphicControl*.

Componentes podem ser de dois tipos no *Delphi*:

- *Visuais* - que aparecem tanto em tempo de projeto quando em tempo de execução, como por exemplo botões. Herdam direta ou indiretamente de *TWinControl* ou *TGraphicControl*;
- *Não-visuais* - não aparecem em tempo de execução, e são derivados direta ou indiretamente de *TComponent*. Um bom exemplo são componentes de acesso a dados, eles não aparecem na aplicação, mas provêem funcionalidade para acessar bancos de dados e se ligam a outros componentes visuais que apresentam esses dados ao usuário.

Mão na Massa

Vamos começar criando um componente não visual, que fará a validação do CPF. Você poderá então utilizar o componente em qualquer aplicação que precisar fazer esse tipo de validação.

Abra o *Delphi 7* e dê um *File>Close All*. Clique *File>New>Other>New>Package* para criar um pacote. No *Delphi*, todos os componentes devem ser instalados dentro de um pacote. Salve esse pacote como *"c:\Easy\pkClubeDelphi.dpk"*. Você verá um editor semelhante ao mostrado na **Figura 2.**

Vamos adicionar um componente nesse pacote. Clique em *Add* e na tela que aparece clique na aba *New Component*, pois vamos criar um novo componente. O primeiro passo é escolher a classe base, como vamos criar um componente não visual, vamos escolher *TComponent*. Em *Class Name* digite *"TCPF"*. Note que as classes no *Delphi* sempre iniciam com *T*, de *type*, apesar de não obrigatória, essa é uma boa prática. Em

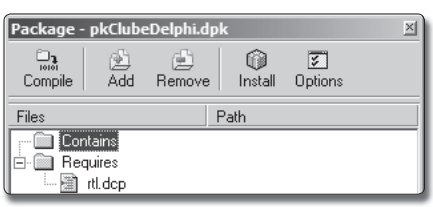


Figura 2. Editor do pacote

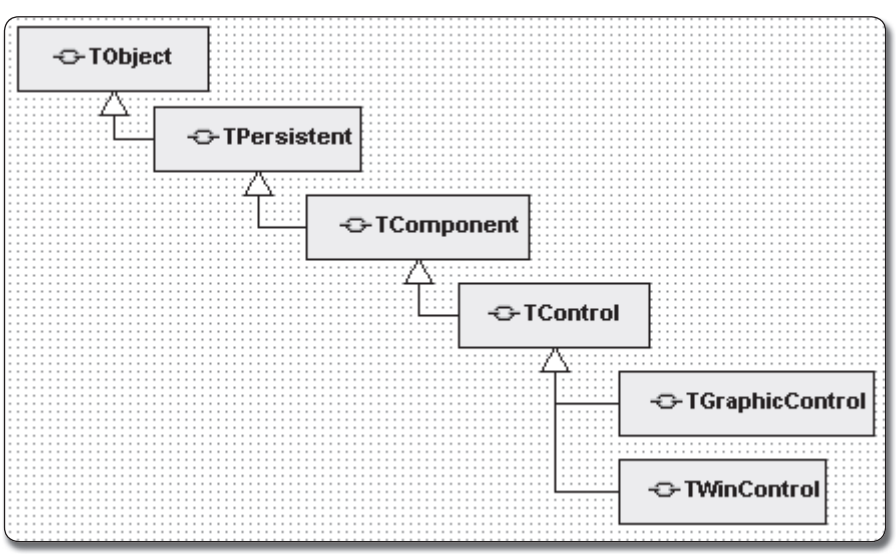


Figura 1. "Core" das classes da VCL

Nota do DevMan

No *Delphi*, termos primitivos como *integer* e *string* não são tratados como objetos. No .net Framework, todos os tipos, mesmo primitivos, são derivados de *Object* (sem o "t"). Isso quer dizer que você pode fazer coisas do tipo:

1.*ToString*();

Pois "1", que é um *integer*, é um objeto, e possui métodos herdados que podem ser chamados.

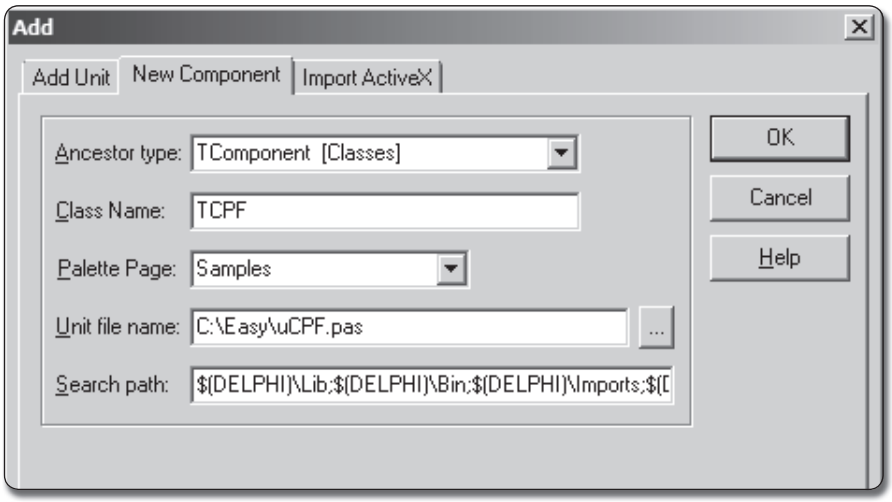


Figura 3. Criando um componente

Listagem 1. Esqueleto da classe

```
unit uCPF;

interface

uses
  SysUtils, Classes;

type
  TCPF = class(TComponent)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('ClubeDelphi', [TCPF]);
end;

end.
```

Listagem 2. Função para validar CPF

```
function TCPF.VerificaCPF(CPF: string): boolean;
var
  StrCpf : string;
  d1,d2,n1,n2,na,loop : integer;
  LocalResult : boolean;
begin
  Localresult:=false;
  StrCpf:=CPF;
  na:=1;
  for loop:=1 to length(StrCPF)-2 do
  begin
     n1:=n1+(11-na)*strtoint(StrCPF[loop]);
     n2:=n2+(12-na)*strtoint(StrCPF[loop]);
     na:=na+1;
  end;
  d1:=0;
  if (n1 mod 11)>=2 then
      d1:=11-(n1 mod 11);
  d2:=0;
  n2:=n2+2*d1;
  if (n2 mod 11)>=2 then
      d2:=11-(n2 mod 11);
  if (inttostr(d1)=StrCPF[10]) and
     (inttostr(d2)=StrCPF[11]) then
      Localresult:=true;
  Result:=LocalResult;
end;
```

Unit file name digite o mesmo caminho do pacote, dando um nome à *unit*, como "*C:\Easy\uCPF.pas*". A **Figura 3** representa esses passos. Clique em Ok.

Neste momento, o *Delphi* criou o esqueleto da classe para nós, que deve estar semelhante ao da **Listagem 1**. Nela temos a seção *uses*, importando as *units* básicas para trabalhar com componentes, e a seguir o principal, a declaração da nossa classe, ou melhor, componente já que herda de *TComponent* como você pode ver no código. Os especificadores de visibilidade estão apenas com comentários, falaremos mais sobre eles adiante. E finalmente, na implementação, a linha que registra o componente e faz ele aparecer no *IDE*. Note que ele será instalado na paleta *Samples*, modifique para "ClubeDelphi".

Codificando o método para verificação do CPF

O nosso componente vai validar um determinado CPF. Mas quem for usar o componente, não quer ter a mínima idéia de como isso funciona. Esse é um dos princípios da *POO*, encapsulamento. Na vida real, quando adquirimos um carro, nós o usamos para passear, viajar, mas não precisamos saber como ele funciona internamente, como é o motor, que peças o compõe etc. Assim é a *POO*.

Então nosso método de verificação deve ser privado, ou seja, declarado na seção *private*, para que somente a classe atual e internamente possa fazer uso dessa verificação. O que vamos expor para o mundo a fora (público) é somente se um CPF é válido ou não. Declare então o seguinte na sessão *private*:

```
private
   function VerificaCPF(CPF: string):
boolean;
```

A função recebe como parâmetro um CPF e retorna verdadeiro ou falso, dependendo se ele é válido ou não. A propósito, comece a chamar a função agora de *método*, pois toda a função ou *procedure* que se encontre dentro de uma classe passa a se chamar *método*. Agora aperte *Shift+Ctrl+C* para que o *Delphi* gere o cabeçalho para implementarmos esse método. Implemente-o conforme a **Listagem 2.** Não vou entrar em detalhes sobre a fórmula da verificação do CPF pois foge ao escopo do artigo.

Criando propriedades

Se instalássemos o componente agora, o desenvolvedor não poderia fazer praticamente nada com ele. Precisamos de uma forma de expor funcionalidade externa para o componente. Isso se consegue através dos modificadores *public* e *published*.

Em nosso exemplo, vamos criar duas propriedades. Mas o que vem a ser uma propriedade? Um carro, por exemplo, tem cor, modelo, ano, quilometragem, isso são atributos (propriedades) do veículo. Da mesma forma, nosso componente vai ter um atributo (propriedade), chamado CPF, que é por onde o usuário do componente irá entrar o número a ser validado.

Uma segunda propriedade vai indicar ao usuário se o CPF que ele entrou é válido ou não (ele não vai chamar o *VerificaCPF*, esse é interno). Então, basta que ele forneça a propriedade, e depois verifique o valor dessa propriedade para ver se o CPF é válido ou não.

Essas propriedades, como precisam ser vistas ao mundo externo, devem ser públicas. E o *Delphi*, tem uma opção especial, se declararmos as propriedades como *published*, elas aparecerão na janela *Object Inspector* do *IDE* e podem ser testadas em tempo de *design*! Vamos ver como fazer isso. Na *unit* da classe, declare as seguintes propriedades:

```
published
  property CPF: string read FCPF write
SetCPF;
  property Valido: boolean read FValido
write FValido;
```

Repare que uma propriedade na verdade não faz nada. Ela possui dois "métodos" chamados *get* e *set*. Em outras palavras, quando atribuímos um valor para a propriedade CPF (*set*), ela delega esse valor a uma função chamada *SetCPF*. Quando lemos a propriedade, ela verifica e devolve o valor que está em *FCPF*. *FCPF* é uma variável auxiliar, de armazenamento, que guarda os dados em memória (propriedades não guardam nada). É uma espécie de mapeamento, o que você fizer em CPF, afeta *FCPF*, ou por meio de um método, ou por leitura/gravação direta na variável *FCPF* (que é privada).

A outra propriedade, *Valido*, indicará se o CPF entrado na outra propriedade é válido. Observe que estamos fazendo o mapeamento direto para uma variável chamada *FValido*. Aperte *Shift+Ctrl+C* para ver o *Delphi* gerar essas variáveis e métodos privados, bem como os cabeçalhos de implementação. Sua *unit* deve estar semelhante à **Listagem 3**.

Implementando a funcionalidade

No momento em que o usuário configurar o valor para a propriedade CPF, será automaticamente disparado o método *SetCPF*. Foi isso que dissemos na declaração da propriedade. E é nesse exato momento que vamos verificar o CPF e atribuir um valor para a propriedade *Valido*, como mostra a **Listagem 4**.

Definindo um ícone para o componente

Nosso componente está pronto, falta apenas definir um ícone para que apareça na paleta de componentes do *Delphi* (ao invés do padrão). Para isso abra o *Image Editor*, que se encontra no menu do *Delphi* do Iniciar do *Windows*.

Clique em *File>New>Component Resource File (.dcr)*. Dê um clique de direita em *Contents* e escolha *New>Bitmap*. Confirme a janela de opções com Ok (escolha *Super Vga* e 24x24). Renomeie o recurso criado

Listagem 3. Unit da classe com propriedades

```
  TCPF = class(TComponent)
  private
    FValido: boolean;
    FCPF: string;
    function VerificaCPF(CPF: string): boolean;
    procedure SetCPF(const Value: string);
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    property CPF: string read FCPF write SetCPF;
    property Valido: boolean read FValido write FValido;
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Samples', [TCPF]);
end;

{ TCPF }

procedure TCPF.SetCPF(const Value: string);
begin
  FCPF := Value;
end;
...
```

Listagem 4. Verificando se o CPF é válido pela propriedade

```
procedure TCPF.SetCPF(const Value: string);
begin
  FCPF := Value;
  if VerificaCPF(FCPF) then
    FValido := true
  else
    FValido := false;
end;

ou se preferir

procedure TCPF.SetCPF(const Value: string);
begin
  FCPF := Value;
  Fvalido := VerificaCPF(FCPF);
end;
```

para o mesmo nome da classe, ou seja, *TCPF*, depois dê um duplo clique nele e desenhe ou cole seu ícone à vontade. Veja como ficou o meu na **Figura 4**. Dica: use o *Ctrl+I* e *U* para dar *zoom in* e *zoom out*. Salve o arquivo como *pkClubeDelphi.dcr* no mesmo diretório do pacote.

Agora, de volta ao IDE, clique em *Project>View Souce*. Vamos dizer ao *Delphi* para compilar nosso recurso de imagem dentro do pacote criado. Para isso adicione a seguinte linha no arquivo *DPK*:

```
package pkClubeDelphi;
{$R pkClubeDelphi.dcr}
...
```

Pronto, agora basta voltar ao editor do pacote e clicar em *Install*. Se tiver dificuldade em instalar pelo editor, poderá usar o *Project>Project Maganer*. Uma mensagem irá aparecer dizendo que o componente foi registrado e agora nosso *TCPF* aparece na paleta (**Figura 5)**.

Listagem 5. Verificando CPF em tempo de execução

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   CPF1.CPF := Edit1.Text;
   if CPF1.Valido then
      Label1.Caption := 'CPF é válido'
   else
      Label1.Caption := 'CPF é inválido';
end;
```

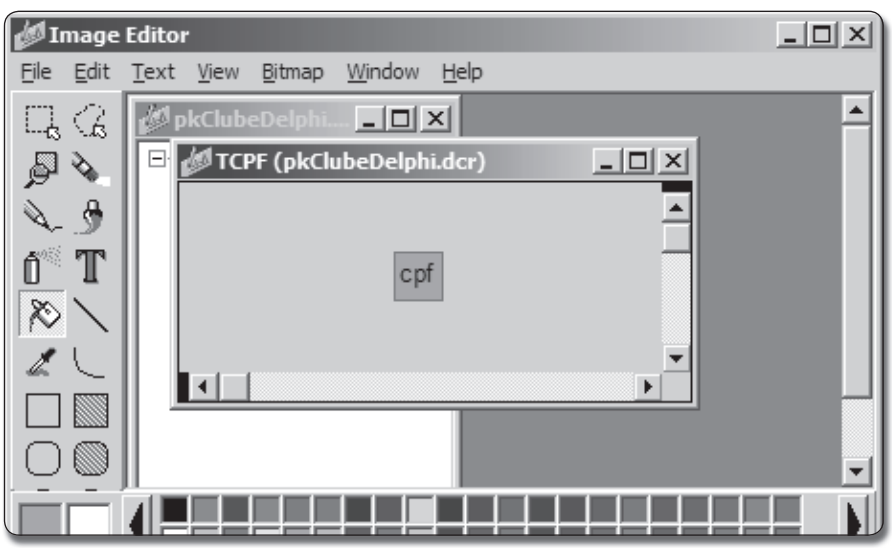


Figura 4. Criando um ícone para o componente

Testando o componente

Você já pode fechar e salvar todos os arquivos e iniciar uma nova aplicação no *Delphi*. Coloque no *form* um *Edit*, um *Button* e um *Label*, além de um componente CPF. Já faça um teste, experimente no *Object Inspector* mesmo selecionar o *CPF1* e digitar um CPF. Observe que a propriedade *Valido* já indicará se ele é válido ou não! (**Figura 6**)

Em tempo de execução, o procedimento é bastante simples. Dê um duplo clique no botão e digite o código da **Listagem 5.** Nesse caso, vamos verificar o CPF digitado no *Edit* quando o usuário clicar no botão, e exibir uma mensagem no *Label*. Observe que basta atribuir a propriedade de CPF do componente e depois verificar a propriedade *Valido* (**Figura 7**).

Conclusão

A criação de componentes se baseia fortemente na *POO*. Por esse motivo, se mostra bastante poderosa. Aliada ao *IDE RAD* do *Delphi*, componentes podem tornar sua vida mais simples, mais robusta e principalmente, mais produtiva. Abraço a todos e sucesso com o *Delphi*. ●

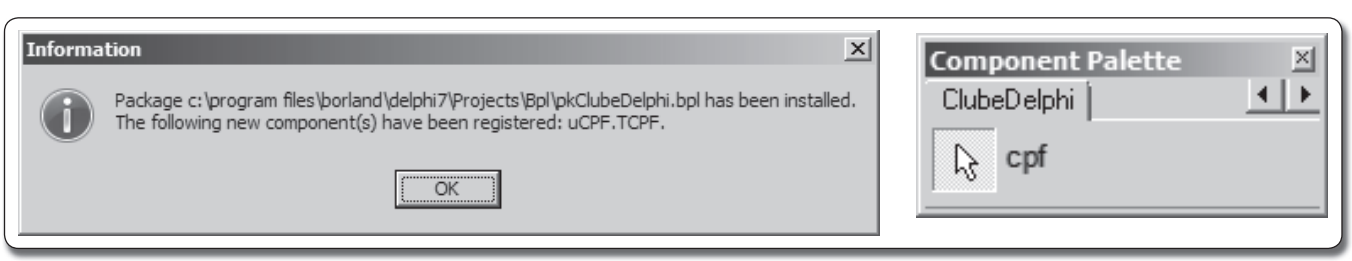


Figura 5. Componente instalado

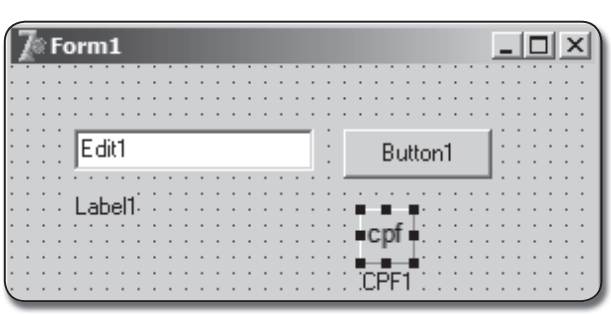


Figura 6. Tela da aplicação

Figura 7. Aplicação em execução

POR QUE A ALOG É LÍDER EM HOSTING GERENCIADO?

- Dois Data Centers próprios – RJ e SP
- Projetos de Hosting customizados
- Foco em serviços de Data Center
- Gerência total dos servidores
- 850 clientes corporativos
- 8 mil m² de área construída
- Destaque do Ano no segmento Internet Serviços no Anuário Telecom 2007

EMPRESA CERTIFICADA ISO 9001 ANAB ACCREDITED BSI

EQUIPE CERTIFICADA EM:

PMI	PMP (PROJECT MANAGEMENT PROFESSIONAL)
ITIL	ITIL FOUNDATION
MICROSOFT	MCP (MICROSOFT CERTIFIED PROFESSIONAL), MCDST (MICROSOFT CERTIFIED DESKTOP SUPPORT TECHNICIAN), MCSE (MICROSOFT CERTIFIED SYSTEM ENGINEER)
CISCO	CCNA (CISCO CERTIFIED NETWORK ASSOCIATE) CCDA (CISCO CERTIFIED DESIGN ASSOCIATE)
RED HAT	RHCE (RED HAT CERTIFIED ENGINEER)
LPI INSTITUTE	LPI-1 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 1) LPI-2 (LINUX PROFESSIONAL INSTITUTE CERTIFIED LEVEL 2)
GIAC	GSEC (GIAC SECURITY ESSENTIALS GOLD)
CHECKPOINT	CCSA (CHECKPOINT CERTIFIED SECURITY ADMINISTRATOR)

ALOG
DATA CENTERS DO BRASIL

www.alog.com.br | 0800 282 3330

Hosting Gerenciado® | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Segurança na Web com PHP

Confira algumas dicas de segurança para evitar que usuários habilidosos invadam seu Web Site

Resumo DevMan

Nesse artigo veremos algumas dicas de segurança para evitar que usuários com um pouco mais de experiência consigam invadir o site e forjar informações.

Veremos como utilizar sessões em login's de forma que o usuário não consiga entrar diretamente em uma página sem fazer autenticação.

Nesse artigo veremos

- Dicas de segurança;
- Utilização de sessões;
- Criação de login para acesso à área de administração.

Qual a finalidade

- Manter o web site protegido evitando possíveis invasões.

Quais situações utilizam esses recursos?

- Todo web site precisa de recursos de segurança para evitar invasões e conseqüentemente problemas futuros.

Adriano Santos
falecom@adrianosantos.pro.br
é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É Editor Técnico, Colunista e Membro da Comissão Editorial das revistas ClubeDelphi e WebMobile. Gerente de Desenvolvimento SoftPark, empresa parceira Borland. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

A internet como um todo cresceu bastante nos últimos tempos e tornou-se um meio de comunicação quase que indispensável no dia-a-dia. Aliás, além de um meio de comunicação, acabou virando "um lugar" onde podemos encontrar todo tipo de conteúdo, tal como: entretenimento, informação, cultura, jornalismo, compra e venda de produtos, sexo, etc. Milhares e milhares de Web Sites surgem a cada dia e são inúmeros os temas discutidos na grande rede.

Junto com a praticidade da WWW, que trouxe solução para diversos problemas, vieram muitos outros como é o caso da segurança. Os golpes, spams, fraudes, invasões, etc. são cada vez mais freqüentes e movimentam milhões de dólares todos os anos. *Crackers, hackers,* enfim, os malfeitores da internet desafiam o tempo todo profissionais e autoridades especializadas em segurança que ganham para manter a integridade de dados de suas redes corporativas. As mais diversas ferramentas e conceitos de segurança são implementados e criados a todo tempo, porém muitos profissionais desconhecem técnicas básicas para

o desenvolvimento de sites seguros e com poucas chances de erro.

O tema desse artigo é justamente alertar sobre alguns cuidados a se tomar para evitar determinados problemas e com isso evitar muitos transtornos no dia-a-dia.

Veremos como evitar o envio de arquivos ou a tentativa de busca das senhas do servidor. Faremos também um pequeno sistema e o adaptaremos para usar sessões evitando o acesso à página sem efetuar login.

Envitando passagem de páginas por parâmetro

A primeira dica é prever que o usuário do seu site nem sempre tem a intenção de entrar com as informações exatamente como é solicitada, ou seja, burlar um site pode ser bastante divertido para alguns e um negócio para outros. Portanto, temos que tomar algumas precauções para evitar que o internauta tenha acesso total aos seus arquivos e/ou senhas.

O primeiro caso está no fato de abrir arquivos baseados na entrada do usuário, por exemplo: quando usamos os métodos *include()* e *require()*, os arquivos informados no método podem ser variáveis e portanto podem vir de informações do usuário. Podemos permitir, por algum motivo, que o usuário informe qual o arquivo ele pode carregar nos métodos *include()* e *require()* através de um formulário ou *QueryString()*, veja:

```
<?php
  include($minha_pagina);
?>
```

No código anterior estamos informando ao método *include()* que uma variável chamada *$minha_pagina* deverá ser incluída na página atual, ou seja, qualquer informação poderá vir nessa variável. Vejamos a **Listagem 1**.

No código da **Listagem 1** criamos um formulário onde o usuário poderá informar qual o arquivo deseja enviar/carregar. Veja a **Figura 1** como criamos o layout dessa página.

Há um enorme perigo em se deixar o usuário carregar arquivos dessa forma. Digamos que o nosso usuário é um pouco mais esperto e resolve fazer um pequeno teste: colocar no campo o endereço do arquivo de senhas do PHP. Normalmente o arquivo de senhas fica alocado em */etc/passwd*, logo podemos incluir essa informação no arquivo solicitado pelo nosso formulário.

O arquivo *validar.php*, que será chamado por essa página, receberá o conteúdo do campo e o executará como segue:

```
include($minha_pagina);
```

Automaticamente o arquivo *passwd* poderá ser carregado na tela e obviamente estará disponível para leitura caso a variável *register_globals* do arquivo *php.ini* (presente no servidor) estiver configurada como *On*. Claro, o responsável pela instalação do PHP no servidor do *DataCenter* jamais esqueceria de uma informação como esta. Mas o mais sensato é fazer o teste antes e evitar erro futuro. Outra forma de se fazer isso seria digitando o caminho da página na barra de endereços e colocar o endereço que desejar na variável *$minha_pagina* assim como podemos ver a seguir:

```
http://www.seusite.com.br/seu_script.
php?minha_pagina=http://www.meusite.com.br/
script_malicioso.php
```

Veja que é perfeitamente possível passar para a variável um arquivo de outro servidor. Lembrando que o método *include()* pode carregar arquivos externos o que pode acarretar um grande problema.

Uma solução para isso é criar validações para que o sistema não receba qualquer arquivo. Veja um exemplo de código da **Listagem 2**.

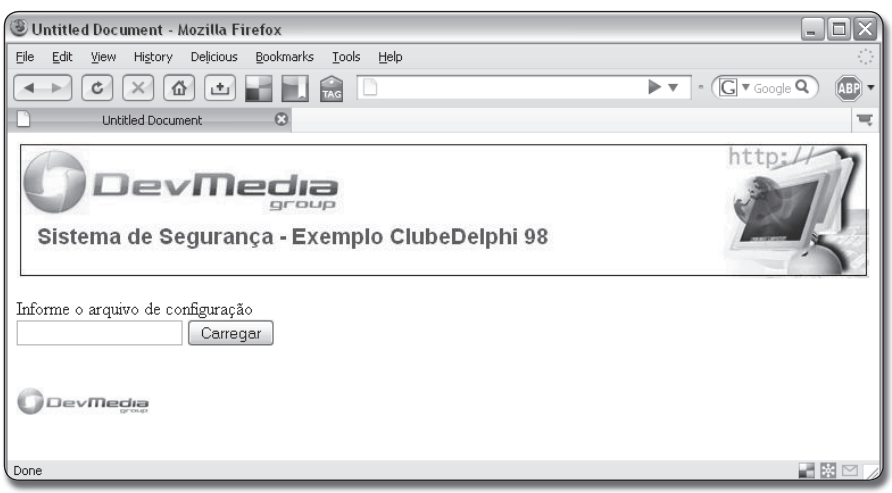


Figura 1. Exemplo de carregamento de arquivo

Listagem 1. Código do arquivo topo.htm

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Untitled Document</title>
</head>
<body>
<img src="topo_artigo.JPG">
<br>
<form action="validar.php" method="post" name="frmValidar" id="frmValidar">
  <p>Informe o arquivo de configura&ccedil;&atilde;o<br>
    <input name="page" type="text" id="page">
    <input name="Carregar" type="submit" id="Carregar" value="Carregar">
  </p>
</form>
<br>
<img src="devmedia.jpg">
</body>
</html>
```

Listagem 2. Exemplo de solução para o problema de carregamento indevido de arquivos

```
<?php
  $paginas = array('index.html','vendas.html','itens.html');
  if (in_array($minha_pagina, $paginas)) {
    include($page);
  }else{
    die("Não é possível carregar esse arquivo.");
  }
?>
```

Veja que agora nós validamos a entrada, ou seja, só os arquivos de nome *index.html, vendas.html* e *itens.html* podem ser carregados. Se tentássemos carregar essa página (*validar.php*) apenas informando um arquivo diferente do que está descrito no *array* de páginas o que obteríamos seria um resultado semelhante à **Figura 2**.

Cuidados com a função eval()

Com a função *eval()*, podemos executar comandos PHP passando uma *string* para ela. Isso mesmo! Podemos incluir código PHP em forma de *string* e executar usando o método *eval()*. Vamos a um exemplo prático. Abra o bloco de notas e digite um código como mostrado na **Listagem 3** e veja que interessante.

Como podemos perceber, utilizamos apenas o método *eval()* e passando para ele uma *string*. Essa *string* contém um comando PHP, texto e, por incrível que pareça, cálculo. Na quinta linha, estamos atribuindo o valor 1 + 2 à variável *$soma*. Em seguida pulamos linha e imprimimos o seu valor. Nas linhas que encontramos o código *HTML
* estamos quebrando linha, ou seja, podemos passar qualquer informação para essa função, até mais código. Veja o resultado desse script na **Figura 3**.

Isso tudo significa que se podemos executar comandos passando-os para um método *eval()* poderemos ter problemas futuramente. Imagine o que aconteceria se passássemos o texto:

```
passthru("cat  /etc/passwd");
```

> **Nota**: A função *passthru()* funciona semelhante à função *exec()*. Com ele podemos executar comandos no PHP também semelhante ao método *eval()*.

Certamente teríamos o arquivo *passwd* impresso em tela, ou seja, todas as senhas definidas. Então é importantíssimo que se use a função *eval()* com muita consciência e nos lugares certos.

Usando register_globals de forma correta

Existe uma propriedade do PHP (*php.ini*) chamada *register_globals*. A idéia dessa propriedade é facilitar a vida do programador, já que as variáveis definidas e atribuídas em um *script* ficarão válidas para toda a aplicação. Mas isso é bastante perigoso, pois caso seu usuário consiga definir um novo valor para uma determinada variável e essa propriedade estiver ativada, ou seja, a opção *ON* (ligado) ele terá domínio da sua aplicação. Em outras palavras terá acesso livre ao seu site sem que você perceba. Veja um exemplo:

```
<?php
  include($minha_pagina);
?>
```

Se o *script* de sua página está definido como o que vimos anteriormente, isso significa que a página está esperando um valor do usuário. Se você esqueceu de fazer a validação em determinadas páginas e o internauta inserir valor na variável *$minha_pagina*, ele passará a ter diversos *podereres* no arquivo e/ou site. Uma forma de fazer isso seria passando para o *script* o valor como a seguir:

```
http://www.suapagina.com.br/seu_script.
php?minha_pagina=http://www.example.com/
maluco.php
```

Listagem 3. Código usando a função eval()

```
<?php
  eval("print \"<br>\";");
  eval("print \"Impresso pelo sistema usando 'eval'\";");
  eval("print \"<br>\";");
  eval("print \"Olá Mundo!\";");
  eval('$soma = 1 + 2'.";");
  eval("print \"<br>\";");
  eval("print \"O valor da soma de 1 + 2 é: $soma\";");
  eval("print \"<br>\";");
?>
```

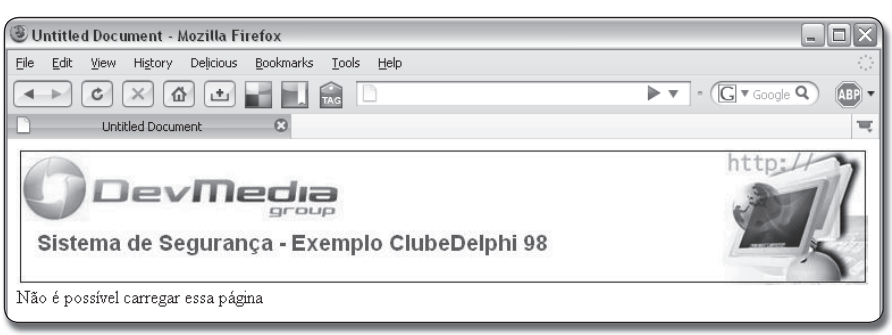


Figura 2. Resultado do teste de validação

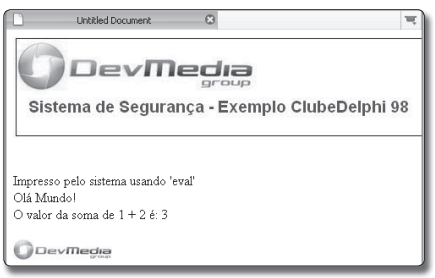


Figura 3. Função eval() em ação

Nota do DevMan

A função *eval()* executa a *string* dada no parâmetro *code_str* como código PHP. Entre outras coisas, isto pode ser útil para guardar código em um campo de texto de um banco de dados para execução posterior.

Há alguns fatores para lembrar quando usar *eval()*. Lembre-se que a *string* passada deve ser código PHP válido, incluindo coisas como terminar os comandos com ponto-e-vírgula assim o compilador não morre uma linha após *eval()*.

Também se lembre que as variáveis que tenham seus valores em *eval()* irão reter estes valores para o resto do *script*.

Um comando *return* irá terminar a execução do *script* imediatamente. No PHP 4, *eval()* retorna NULL a menos que *return()* seja chamado no código executado, caso o valor passado para *return()* é retornado. No PHP 3, *eval()* não retorna um valor.

```
<?php
 $string = 'taça';
 $name = 'café';
 $str = 'Esta é uma $string com o meu $name nela.';
 echo $str."\n";
 eval("\$str = \"$str\";");
 echo $str ."\n";
?>
```

O exemplo acima irá mostrar

Esta é uma $string com o meu $name nela.

Esta é uma taça com o meu café nela.

Fonte: *www.php.net*

Perceba a gravidade do problema. O usuário poderá mandar um arquivo próprio no *script*, com isso ter acesso ao servidor. Uma dica é desenvolver sempre com a propriedade *register_globlas* igual a *Off*, desligada. Além disso, procure ativar todos os erros da aplicação usando o código a seguir na primeira linha de cada arquivo PHP:

```
error_reporting(E_ALL);
```

Com isso ativamos o modo de "impressão" em tela de todos os erros que ocorrem durante o ciclo de vida da aplicação, o que é bastante útil principalmente durante os testes da aplicação.

Validando usuários e administradores

Validar usuários no PHP é muito simples e rápido. A pergunta mais freqüente com certeza é: como autenticar usuários? Nós vamos fazer um exemplo bem simples para fazer com que o usuário tenha acesso a todo o site somente mediante usuário e senha. Veremos que criar uma tela de login e fazer a autenticação não é nada de outro mundo.

Para iniciar nosso projeto crie um novo arquivo no bloco de notas chamado *login.php* e digite o código da **Listagem 4.**

O código da listagem anterior resultará em uma tela semelhante à **Figura 4**. Criei uma imagem de 750px de largura por 120px de altura. Usei algumas imagens da internet e do site da DevMedia. O restante é HTML puro. Na linha a seguir estamos definindo que o formulário será validado no arquivo *administracao.php*:

```
<form action="administracao.php"
method="post" name="frmLogin"
```

Inclui uma tabela (*<table>*) e um formulário (*<form>*) com dois controles *TextBoxes* dentro dele. Coloquei os nomes de *txtLogin* e *txtSenha* e a *action* (ação) do formulário apontei para um arquivo chamado *administracao.php* que ainda criaremos, como já mencionado. Esse arquivo fará a autenticação e nos mostrará um link para o suposto cadastramento de notícias do site ou uma mensagem de *Login e senha inválidos*. Salve o arquivo atual como *login.php* e crie um novo arquivo em branco.

Agora vamos criar o arquivo *administracao.php*. Para agilizar a criação da página e ficarmos com uma tela padronizada, vamos copiar parte do código do arquivo *login.php* e colar em *administracao.php*. O trecho que vamos copiar é a parte que diz respeito ao cabeçalho e rodapé. Assim já teremos as imagens sendo carregadas no site. Desprezaremos então apenas o trecho referente ao formulário de login. Nosso código copiado ficará como na **Listagem 5.**

Perceba que coloquei propositalmente um comentário praticamente no meio do

Listagem 4. Código da tela de login

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Login Teste</title>
</head>
<body>
<center>
<p><img src="topo_artigo.JPG">
</p>
<form action="administracao.php" method="post" name="frmLogin" id="frmLogin">
<table width="750" border="0">
  <tr bgcolor="#009999">
    <td colspan="2">
       <div align="center">
         <font face="Verdana" color="#FFFFFF" size="2">
           <strong>Entre com seus dados para login</strong>
        </font>
       </div>
     </td>
  </tr>
  <tr>
    <td>
      <div align="right">
        <font face="Verdana" color="#000000" size="2">
          Usu&aacute;rio
       </font>
      </div>
    </td>
    <td>
      <div align="left">
        <font face="Verdana" color="#000000" size="2">
          Senha
       </font>
      </div>
    </td>
  </tr>
  <tr>
    <td>
      <div align="right">
        <input name="txtLogin" type="text" id="txtLogin">
      </div>
    </td>
    <td>
      <div align="left">
        <input name="txtSenha" type="password" id="txtSenha">
      </div>
    </td>
  </tr>
  <tr>
    <td>
      <div align="right">
        <input name="btnEntrar" type="submit" id="btnEntrar" value="Entrar">
      </div>
    </td>
    <td>
      <div align="left">
        <input name="btnLimpar" type="reset" id="btnLimpar" value="Limpar">
      </div>
    </td>
  </tr>
</table>
</form>
<table width="750" border="0">
  <tr>
    <td width="175">
      <img src="devmedia.jpg">
    </td>
    <td width="565">
      <font face="Verdana" color="#000000" size="2">
        Todos os direitos reservados para DevMedia Editora
      </font>
     </td>
  </tr>
</table>
<p> </p><p> </p>
</body>
</html>
```

código *HTML* da **Listagem 5**. É na região desse comentário que vamos incluir nosso código de validação e mostrar a mensagem *Login ou senha inválidos* ou incluir um link para cadastramento de notícias.

É realmente bastante comum em PHP fazer a mesclagem de código *HTML, CSS, JavaScript* e o próprio PHP em um mesmo arquivo. Porém poderíamos criar dois novos arquivos para servirem de cabeçalho e rodapé e incluí-los no arquivo PHP usando o método *include()*, assim como fizemos em outros artigos. Isso diminui radicalmente a quantidade de código e ainda mantém um padrão no site. Além disso, existem *frameworks* que auxiliam na separação de código e layout.

Voltando ao nosso exemplo, vamos agora criar o código para validação do usuário e senha. Inclua na linha do comentário já mencionado antes, o código da **Listagem 6**.

Não há segredos, apenas verificamos se as *strings* recebidas nas variáveis *txtLogin* e *txtSenha* batem com o sistema, que nesse caso serão sempre usuário igual a *devmedia* e a senha *1234*. Em seguida imprimimos uma série de *tags
* para forçar a quebra de várias linhas e a por fim a mensagem *Usuário e senha inválidos* em caso negativo. Em caso positivo, ou seja, o usuário e senha digitado são válidos, quebramos novamente algumas linhas e imprimimos a mensagem *Olá ????? seja bem vindo a área de administração do nosso site*, onde as interrogações são o nome do login digitado na tela anterior, ou seja, *devmedia*. Quebramos a linha e imprimimos um link para a página *cad_noticias.php*.

Quando um formulário é enviado para outra página, todos os controles de tela tais como *TextBoxes, ComboBoxes, TextAreas* são enviados como variáveis. Por isso fazemos referência a eles usando o caractere *$* junto com o nome dado ao controle, assim como podemos ver: *$txtLogin* e *$txtSenha*.

Nesse momento poderíamos muito bem fazer uma conexão com o banco de dados e efetuar uma consulta para ver se o usuário digitado existe no banco de dados. Um exemplo simples de consulta pode ser vista na **Listagem 7.**

Veja que é simples. O segredo está nas duas primeiras linhas onde conectamos ao servidor do banco, nesse caso *MySQL*(*mysql_connect*) e selecionamos qual banco de dados queremos usar (*mysql_select_db*). As instruções mostradas variam de banco para banco, veja alguns exemplos na **Tabela 1**.

Agora criaremos apenas mais um arquivo, o *cad_noticias.php*. Repita os passos efetuados anteriormente e crie esse arquivo. Dessa vez o código PHP que usaremos será ainda mais simples, veja:

```
<?
print "<br><br><br>";
print "Área de cadastramento de notícias do
site.";
print "<br><br><br>";
?>
```

Apenas quebramos algumas linhas e

Figura 4. Exemplo de tela de login

Listagem 5. Código do script da tela de administração

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
  charset=iso-8859-1">
<title>Login Teste</title>
</head>
<body>
<center>
<p><img src="topo_artigo.JPG">
</p>

<!- Aqui vai o código PHP para validação da página -->

<table width="750" border="0">
  <tr>
    <td width="175">
      <img src="devmedia.jpg">
    </td>
    <td width="565">
      <font face="Verdana" color="#000000" size="2">
        Todos os direitos reservados para DevMedia Editora
      </font>
    </td>
  </tr>
</table>
<p> </p>
<p> </p>
</body>
</html>
```

Listagem 6. Código de validação

```
<?
  if(!($txtLogin == "devmedia" && $txtSenha == "1234")){
     print "<br><br><br><br><br><br>";
     print "<center>";
     print "<font face='Verdana' color='#000000' size='2'>";
     print "Usuário ou Senha inválido.";
     print "</font>";
     print "<br><br><br>";
  }else{
     print "<br><br><br>";
     print "<font face='Verdana' size='2'>";
     print "Olá <b>" . $txtLogin .
       "</b> seja bem vindo a área de administração do nosso site.";
     print "<br>";
     print "<a href='cad_noticias.php'>Cadastrar Notícias</a>";
     print "</font>";

     print "<br><br><br>";
  }
?>
```

imprimimos um texto informando que estamos na área de cadastramento de notícias. Então, em tese, finalizamos nosso exemplo e podemos testar. Abra o browser então e experimente abrir o arquivo *login.php* e digitar uma senha ou usuário inválido. Sua tela deverá se assemelhar com a **Figura 5** ou com a **Figura 6** caso tenha digitado o usuário e senha válido. E caso tenha clicado no link *Cadastrar Notícias* terá sido redirecionado para a tela de mesmo nome.

Até aqui tudo tranqüilo, nada de especial. Mas vamos ao que interessa, a segurança. Experimente voltar ao login ou mesmo fechar o browser e abrir de novo. Dessa vez digite diretamente na barra de endereços o endereço da página *cad_noticias.php*, algo como *www.seusite.com.br/cad_noticias.php*. Veja que a página se abre tranquilamente sem qualquer validação. É nesse ponto que precisamos nos preocupar, como fazer com que a tela não seja aberta sem validação?

Faremos algumas modificações no código das páginas *administracao.php* e *cad_noticias.php* fazendo com que elas se adaptem ao uso de *sessions*. Abra o arquivo *administracao.php* e adicione o código a seguir antes de qualquer código *HTML*:

```
<?php
  session_start();
?>
```

Nota do DevMan

Uma observação a se fazer em relação ao teste de login e senha. Primeiro perceba como funciona a sintaxe de condicionais no PHP. Usamos o caractere exclamação (!) para negar uma condicional equivalente ao not e o caractere e comercial (&) duas vezes para adicionar uma nova condição, ou seja, é equivalente ao and em Delphi Language. *Portanto, em Delphi nossa condicional ficaria assim:*

if not (txtLogin.Text = 'devmedia') and (txtSenha.Text = '1234')

Outras particularidades da linguagem são os símbolos para atribuição e comparação. Em PHP quando precisamos comparar um valor com outro usamos dois caracteres igual (=) e quanto queremos atribuir um conteúdo a algo usamos apenas um. Veja:

// Exemplo de atribuição

$variavel_nome = 'Adriano Santos';

// Exemplo de comparação

if ($variavel_nome == 'Adriano Santos'){

Desse modo estamos "dizendo" ao *script*, grosseiramente falando, que vamos trabalhar com sessões nessa página. A partir de agora, todos os *scripts* que necessitarem de login, ou seja, façam parte da administração do site ou áreas que não podem estar acessíveis a qualquer usuário precisam ter essa linha incluída antes de qualquer coisa. Caso as primeiras linhas do *script* sejam código PHP a chamada a esse método tem que, obrigatoriamente, ser a primeira linha do contrário uma mensagem será enviada ao usuário da página indicando que a sessão já foi enviada anteriormente, veja:

Warning: session_start() [function.session-start]: Cannot send session cookie - headers already sent by (output started at /home/seusite/www/administracao.php:13) in **/home/seusite/www/administracao.php** on line

Nesse caso o aviso foi enviado pelo PHP porque coloquei a chamada à função *session_start()* logo após a checagem do usuário e senha, ou seja, depois que página já havia sido iniciada. Portanto, em resumo, sempre inicie qualquer página PHP com a chamada ao *session_start()*. Após incluir essa linha logo no início da página *administracao.php*, vá até o código PHP da mesma página e inclua a seguinte linha no *else* da checagem de usuário e senha. Veja o código completo da **Listagem 8**:

```
$_SESSION['login'] = $txtLogin;
```

Veja que estamos atribuindo o usuário digitado no campo *txtLogin* a uma sessão cujo nome é *login*. Isso fará com que o nome digitado permaneça gravado no servidor e conseqüentemente criará uma sessão com o nome proposto. Mais tarde poderemos fazer a verificação em cada página para verificar se a sessão existe ou mesmo imprimir o nome gravado nela.

Abra agora o arquivo *cad_noticias.php* e adicione o código a seguir logo no início da página, antes de qualquer código HTML ou PHP:

```
<?php
  session_start();
  if (!($_SESSION['login'])){
    header("Location: login.php");
  }
?>
```

Listagem 7. Exemplo de validação em banco de dados

```
<?
  $conexao = mysql_connect('localhost', 'usuario_adm', 'senha');
  mysql_select_db('banco_de_dados');
  $sql = 'select USUARIO, SENHA from usuarios where usuario = $txtLogin';
  $resultado = mysql_query($sql);
  $record_count = mysql_num_rows($resultado);
  if ($record_count > 0){
    print "Usuário não encontrado";
  }elseif ($record_count > 0){
    print "Usuário autenticado";
  }
?>
```

Comando	MySQL	PostgreSQL	Interbase/Firebird
Conectar-se ao host	mysql_connect	pg___connect	ibase_connect
Selecionar o banco	mysql_select_db	---	ibase_select_db
Efetuar uma SQL	mysql_query	pg_query	ibase_query
Fechar	mysql_close	pg_close	ibase_close

Tabela 1. Diferença de comandos entre bancos

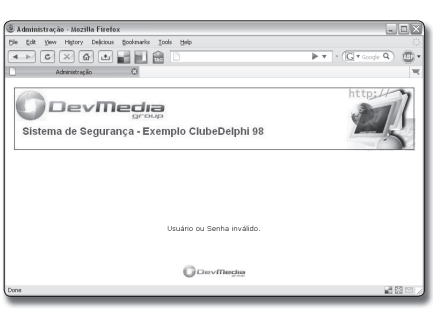


Figura 5. Usuário ou Senha inválido

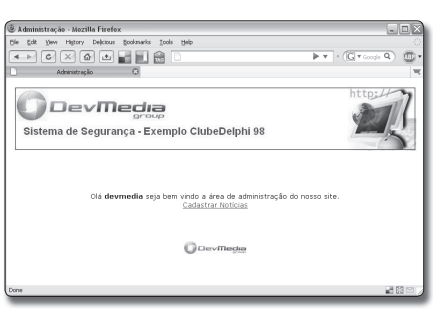


Figura 6. Usuário e senha validado

Observe com calma o trecho de código. Perceba que a primeira linha é justamente a chamada ao *session_start()*. Novamente informo, essa tem que ser sempre a primeira chamada de qualquer arquivo PHP. Em seguida fazemos uma checagem para ver se a sessão *login* existe, caso contrário o usuário é redirecionado para a página *login.php*. Antes de testar a página, faça um último ajuste. Altere o código PHP no corpo dessa página para que fique como a seguir:

```
<?
  print "<font face='Verdana' size='2'>";
  print "<b>" . $_SESSION['login'] . "</b>
  você está logado agora.";
  print "</font>";
  print "<br><br><br>";
  print "Área de cadastramento de notícias do site.";
  print "<br><br><br>";

?>
```

Perceba que já incluí três novas linhas nesse código. A segunda linha faz a impressão do conteúdo da sessão login, ou seja, o nome do usuário logado no site. As linhas acima e abaixo apenas fazem a formatação do texto.

Agora chegou a hora do teste. Abra o browser e tente acessar a página *cad_noticias.php* diretamente pela url, ou seja, digitando na barra de endereços. Veja que somos redirecionados automaticamente para a página de login. Entre como o usuário e senha normalmente e clique no link para nossa página de notícias. Note que a página é carregada normalmente e ainda mostra o nome do usuário logado (**Figura 7**).

Quanto tempo levou para fazer essa adaptação? Talvez bem menos do que levaríamos para voltar o backup de nosso site se alguém conseguisse acessar a página de notícias ou qualquer outra área de administração de nosso site. Sem contar com alguns transtornos que enfrentaríamos.

Listagem 8. Código completo da checagem de usuário e senha

```
<?
  if(!($txtLogin == "devmedia" && $txtSenha == "1234")){
     print "<br><br><br><br><br><br>";
     print "<center>";
     print "<font face='Verdana' color='#000000' size='2'>";
     print "Usuário ou Senha inválido.";
     print "</font>";
     print "<br><br><br>";
  }else{
     $_SESSION['login'] = $txtLogin;
     print "<br><br><br>";
     print "<font face='Verdana' size='2'>";
     print "Olá <b>" . $txtLogin .
       "</b> seja bem vindo a área de administração do nosso site.";
     print "<br>";
     print "<a href='cad_noticias.php'>Cadastrar Notícias</a>";
     print "</font>";
     print "<br><br><br>";
  }
?>
```

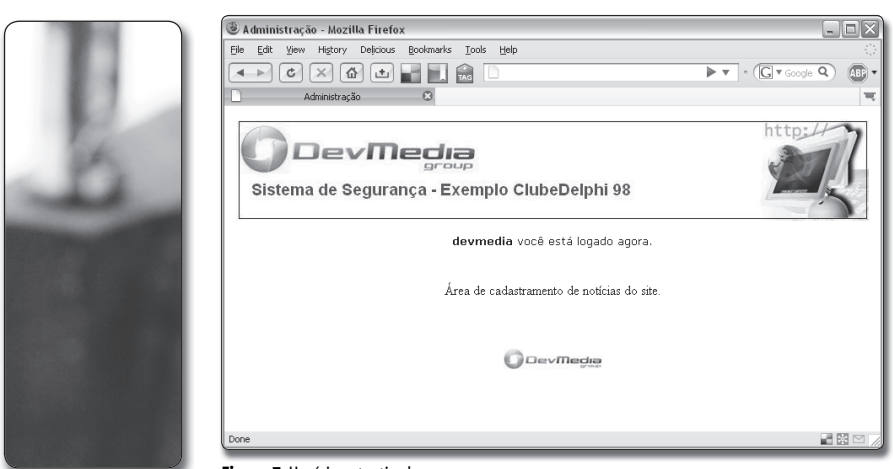


Figura 7. Usuário autenticado

Sem dúvida o tema é polêmico e extenso o bastante para falarmos horas e horas. Essas são apenas algumas dicas do dia-a-dia. O ideal é estudar a fundo tudo que possa ajudar a proteger nossos sites e o de nossos clientes. Segurança na internet é algo importantíssimo.

Considerações finais

Além do item segurança, as sessões são úteis para uma série de outras situações, como por exemplo, em um site de comércio eletrônico. O carrinho de compras desses sites pode ser totalmente trabalhado em cima de sessões, assim podemos armazenar o que o internauta escolheu para comprar. Não usando sessões corremos o risco de enxergar as compras de outros internautas. As sessões são individuais, ou seja, quando definimos que a sessão login iria conter o nome do campo *txtLogin* (*devmedia*) significa que uma sessão foi aberta para esse usuário. Se outro usuário entrasse no site com outro nome, obviamente outra sessão seria iniciada guardando o nome do novo internauta.

Eu recomendo ler o tutorial sobre sessões no site do PHP em *www.php.net/manual/pt_BR/intro.session.php*. Há uma enorme quantidade de explicações sobre o assunto.

Conclusão

Segurança na internet é um dos fatores determinantes para o sucesso de um site e um dos assuntos mais discutidos. Aqui vimos apenas algumas técnicas, porém o assunto é bem mais complexo e exige a leitura de livros especializados para melhorar ainda mais a segurança de seu site. Recomendo que adquira livros, participe de fóruns e palestras para aumentar ainda mais seu conhecimento.

Um forte abraço e até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Conhecimento faz diferença!

Mais de 60 mil downloads na primeira edição!

Faça já sua assinatura digital! | *www.devmedia.com.br/es*

Faça um *up grade* em sua carreira.

Em um mercado cada vez mais focado em qualidade ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lança para os desenvolvedores brasileiros sua primeira revista digital totalmente especializada em Engenharia de Software. Todos os meses você irá encontrar artigos sobre Metodologias Ageis; Metodologias tradicionais (*document driven*); ALM (*application lifecycle management*); SOA (aplicações orientadas a servicos); Analise de sistemas; modelagem; Métricas; orientação à objetos; UML; testes e muito mais. **Assine já!**

DevMedia GROUP

A INVASÃO DE PROPRIEDADE NÃO SE LIMITA A RESIDÊNCIAS.

Assegure-se que seu software está protegido pelo líder em segurança da informação, antes que seja tarde demais.

SafeNet tem à sua disposição a proteção de software mais avançada e segura. As Chaves de Hardware Sentinel oferecem tecnologia que incorpora criptografia de chave pública, criptografia AES e autenticação interna para garantir o mais alto nível de proteção anti-pirataria.

Peça seu Kit de Desenvolvimento hoje mesmo.

Entre em contato em +55 11 4208-7700 em SP-Brasil
ou por e-mail infopirataria@safenet-inc.com

25 YEARS
1983–2008

SafeNet